Computer Network Security

COMP 178 | Spring 2025 | University of the Pacific | Jeff Shafer

Cryptographic Libraries

(And Lab 12!)

Let’s start using cryptography in our programs!

Such as Lab 12…

Hybrid Cryptography

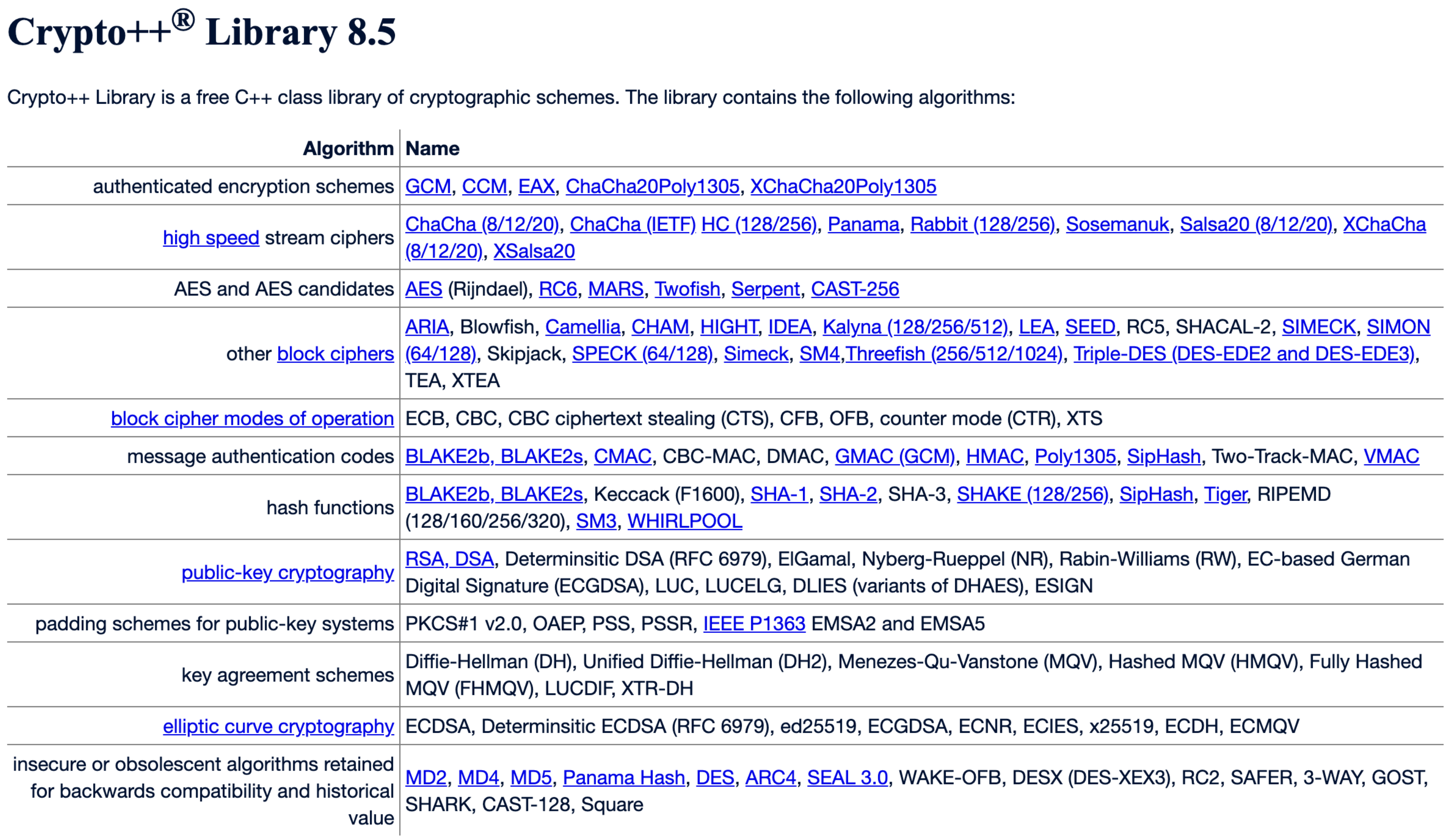
- Take message and encrypt with random symmetric key
- Take symmetric key and encrypt with asymmetric public key of recipient
- Security of asymmetric key exchange ✓
- Performance of symmetric encryption ✓

What library should we use to accomplish this?

Hmmmn, let's search for crypto library…

Google

How does this library look?



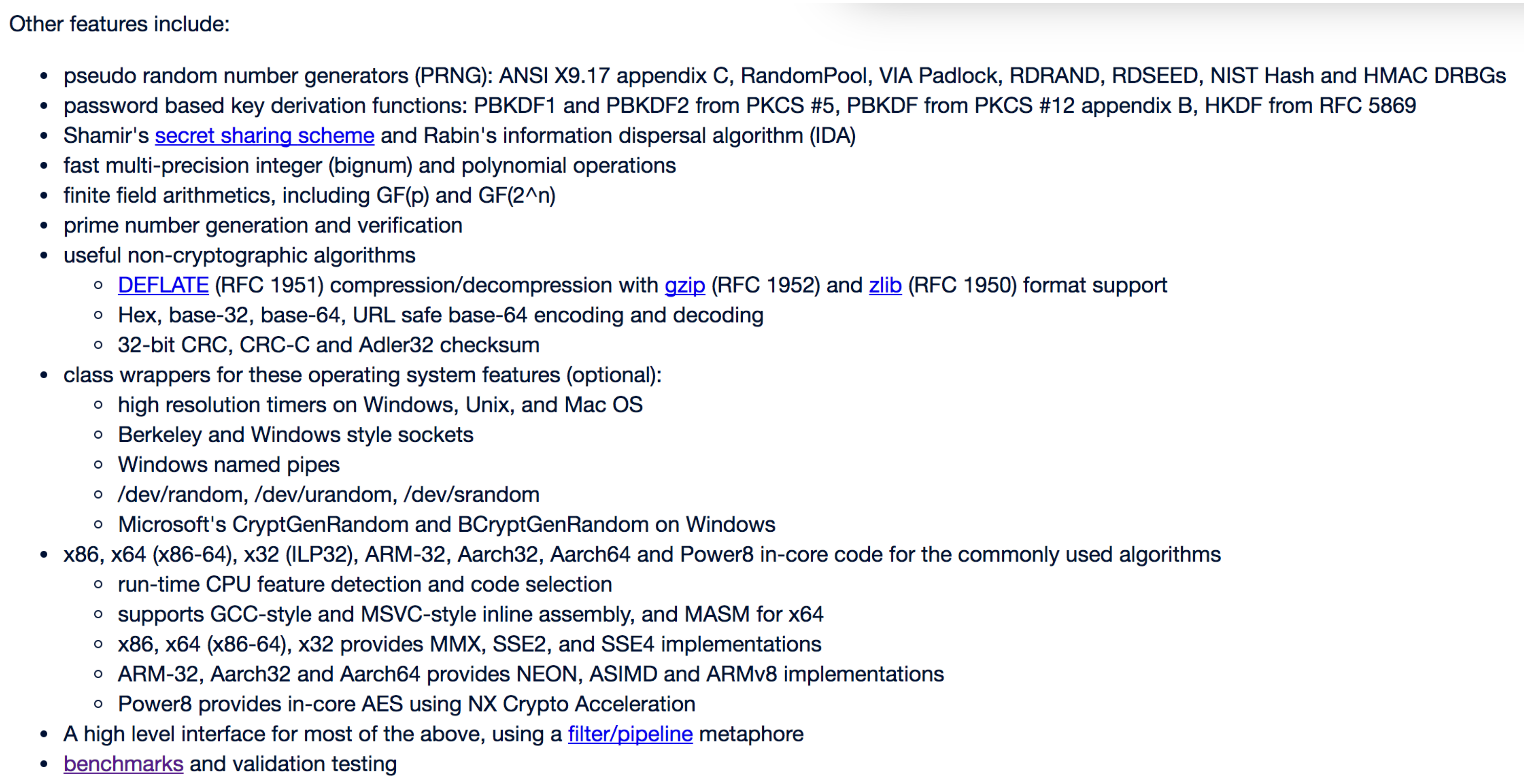
Crypto++® Library 8.5

Crypto++ Library is a free C++ class library of cryptographic schemes. The library contains the following algorithms:

Algorithm	Name
authenticated encryption schemes	GCM, CCM, EAX, ChaCha20Poly1305, XChaCha20Poly1305
high speed stream ciphers	ChaCha (8/12/20), ChaCha (IETF) HC (128/256), Panama, Rabbit (128/256), Sosemanuk, Salsa20 (8/12/20), XChaCha (8/12/20), XSalsa20
AES and AES candidates	AES (Rijndael), RC6, MARS, Twofish, Serpent, CAST-256
other block ciphers	ARIA, Blowfish, Camellia, CHAM, HIGHT, IDEA, Kalyna (128/256/512), LEA, SEED, RC5, SHACAL-2, SIMECK, SIMON (64/128), Skipjack, SPECK (64/128), Simeck, SM4,Threefish (256/512/1024), Triple-DES (DES-EDE2 and DES-EDE3), TEA, XTEA
block cipher modes of operation	ECB, CBC, CBC ciphertext stealing (CTS), CFB, OFB, counter mode (CTR), XTS
message authentication codes	BLAKE2b, BLAKE2s, CMAC, CBC-MAC, DMAC, GMAC (GCM), HMAC, Poly1305, SipHash, Two-Track-MAC, VMAC
hash functions	BLAKE2b, BLAKE2s, Keccack (F1600), SHA-1, SHA-2, SHA-3, SHAKE (128/256), SipHash, Tiger, RIPEMD (128/160/256/320), SM3, WHIRLPOOL
public-key cryptography	RSA, DSA, Determinsitic DSA (RFC 6979), ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), EC-based German Digital Signature (ECGDSA), LUC, LUCELG, DLIES (variants of DHAES), ESIGN
padding schemes for public-key systems	PKCS#1 v2.0, OAEP, PSS, PSSR, IEEE P1363 EMSA2 and EMSA5
key agreement schemes	Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), Hashed MQV (HMQV), Fully Hashed MQV (FHMQV), LUCDIF, XTR-DH
elliptic curve cryptography	ECDSA, Determinsitic ECDSA (RFC 6979), ed25519, ECGDSA, ECNR, ECIES, x25519, ECDH, ECMQV
insecure or obsolescent algorithms retained for backwards compatibility and historical value	MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK, CAST-128, Square

https://www.cryptopp.com/

How does this library look?



Other features include:

- pseudo random number generators (PRNG): ANSI X9.17 appendix C, RandomPool, VIA Padlock, RDRAND, RDSEED, NIST Hash and HMAC DRBGs
- password based key derivation functions: PBKDF1 and PBKDF2 from PKCS #5, PBKDF from PKCS #12 appendix B, HKDF from RFC 5869
- Shamir's secret sharing scheme and Rabin's information dispersal algorithm (IDA)
- fast multi-precision integer (bignum) and polynomial operations
- finite field arithmetics, including GF(p) and GF(2^n)
- prime number generation and verification
- useful non-cryptographic algorithms
  - DEFLATE (RFC 1951) compression/decompression with gzip (RFC 1952) and zlib (RFC 1950) format support
  - Hex, base-32, base-64, URL safe base-64 encoding and decoding
  - 32-bit CRC, CRC-C and Adler32 checksum
- class wrappers for these operating system features (optional):
  - high resolution timers on Windows, Unix, and Mac OS
  - Berkeley and Windows style sockets
  - Windows named pipes
  - /dev/random, /dev/urandom, /dev/srandom
  - Microsoft's CryptGenRandom and BCryptGenRandom on Windows
- x86, x64 (x86-64), x32 (ILP32), ARM-32, Aarch32, Aarch64 and Power8 in-core code for the commonly used algorithms
  - run-time CPU feature detection and code selection
  - supports GCC-style and MSVC-style inline assembly, and MASM for x64
  - x86, x64 (x86-64), x32 provides MMX, SSE2, and SSE4 implementations
  - ARM-32, Aarch32 and Aarch64 provides NEON, ASIMD and ARMv8 implementations
  - Power8 provides in-core AES using NX Crypto Acceleration
- A high level interface for most of the above, using a filter/pipeline metaphore
- benchmarks and validation testing

https://www.cryptopp.com/

So many options!!

More options is good, right?

Library Primitives

- How to accomplish hybrid cryptography with a traditional (low-level) library
  - Choose algorithms and parameters, e.g. AES 256 bit, RSA 4096 bit etc.
  - Generate RSA key pair
  - Generate random AES key and nonce
  - Use AES key to encrypt data
  - Hash encrypted data
  - Read RSA private key from wire format
  - Use key to sign hash
  - Read recipient’s public key from wire format
  - Use public key to encrypt AES key and signature
- Many parameters and options to select along the way!

“Crypto is Broken or How to Apply Secure Crypto as a Developer”
https://blog.codecentric.de/en/2014/03/crypto-broken-apply-secure-crypto-developer/

Developers 101

From Hacker News thread on cryptography

Question:
"What is wrong with mcrypt?"

(One of several PHP libraries for encryption)

Answer:
It's a low-level crypto library that leaves avoidance of virtually all the exploitable crypto mistakes as an exercise for the programmer. 😜

https://paragonie.com/blog/2015/05/if-you-re-typing-word-mcrypt-into-your-code-you-re-doing-it-wrong

Developers 101

You should never type A… E… S…. into your code anywhere!

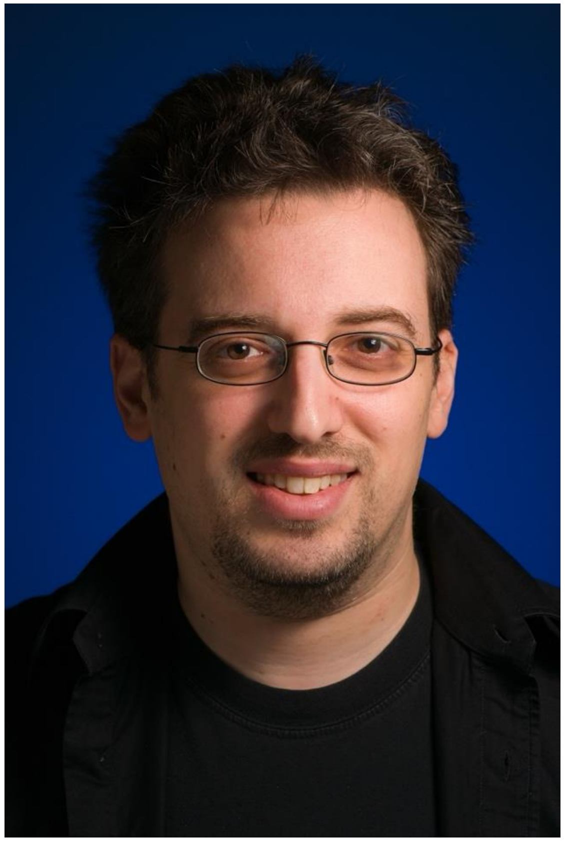
And you should <u>really never</u> type D…E…S… into your code

And you should think twice before type M..D…5 or S..H..A… as well

Robert McCall, “The Prologue and the Promise”

NaCL – our Utopia?

NaCL



- **N**ot **A**nother **C**rypto **L**ibrary (or "Salt")
  - https://nacl.cr.yp.to/
- Released by Daniel J. Bernstein (DJB) in 2011
  - Mathematician and cryptographer
  - Research professor at University of Illinois at Chicago
  - https://cr.yp.to/djb.html
  - He's like the "Richard Stallman" (GNU Founder) of cryptography

Bernstein v. United States (1996)

While a graduate student at the University of California at Berkeley, Bernstein completed the development of an encryption equation (an "algorithm") he calls "Snuffle." Bernstein wishes to publish (a) the algorithm (b) a mathematical paper describing and explaining the algorithm and (c) the "source code" for a computer program that incorporates the algorithm. Bernstein also wishes to discuss these items at mathematical conferences, college classrooms and other open public meetings. The Arms Export Control Act and the International Traffic in Arms Regulations (the ITAR regulatory scheme) required Bernstein to submit his ideas about cryptography to the government for review, to register as an arms dealer, and to apply for and obtain from the government a license to publish his ideas. Failure to do so would result in severe civil and criminal penalties. Bernstein believes this is a violation of his First Amendment rights and has sued the government.

https://www.eff.org/cases/bernstein-v-us-dept-justice

Bernstein v. United States (1996)

- Ruling by 9th Circuit Court of Appeals

Software source code is speech protected by the First Amendment and government regulations preventing its publication were unconstitutional

"This court can find no meaningful difference between computer language, particularly high-level languages as defined above, and German or French....Like music and mathematical equations, computer language is just that, language, and it communicates information either to a computer or to those who can read it..."

-Judge Patel, April 15, 1996

https://www.eff.org/deeplinks/2015/04/remembering-case-established-code-speech

NaCL Properties

- **Expert selection of default primitives**

  Typical cryptographic libraries force the programmer to specify choices of cryptographic primitives: e.g., “sign this message with 4096-bit RSA using PKCS #1 v2.0 with SHA-256.”

  Most programmers using cryptographic libraries are not expert cryptographic security evaluators. ☹

  Often programmers pass the choice along to users—who usually have even less information about the security of cryptographic primitives. ☹

https://nacl.cr.yp.to/features.html

NaCL Properties

- High-level primitives instead of low-level operations
  - Tiny number of functions!
- High-speed implementation
- Automatic CPU-specific tuning
- Resistant to side-channel timing attacks
  - No data-dependent branches
  - No data-dependent array indices
  - No dynamic memory allocation

https://nacl.cr.yp.to/features.html

Challenges

- Implementation not portable/cross-platform
- Implementation is not a shared library
- Implementation difficult to package due to build system and compilation requirements
- *System designed as a research exercise, instead of for programmers*

Libsodium

Libsodium

- Cross-platform fork of NaCL with API bindings for common programming languages beyond C/C++
  - http://www.libsodium.org
  - https://github.com/jedisct1/libsodium
- Uses same implementation of crypto primitives as NaCL
- Passed security audit
  - https://www.privateinternetaccess.com/blog/2017/08/libsodium-audit-results/

Libsodium Features

- Authenticated public-key encryption
- Authenticated shared-key encryption *(symmetric)*
- Hashing / keyed hashing
- Cryptographically secure PRNG

Libsodium Algorithm – Public Key

- **Asymmetric encryption: Curve25519**
- Elliptic curve Diffie-Hellman key agreement (X25519)
  - Why?
    - Not patent encumbered
    - No “secret constants” that were “helpfully” suggested by the NSA with no documentation on why they were selected
  - Used where?
    - https://ianix.com/pub/curve25519-deployment.html
    - Libsodium, OpenSSL, LibreSSL, libssh, …
    - Standard in TLS 1.3
    - OpenSSH, iOS, Signal messenger, WhatsApp, WireGuard VPN

Libsodium Algorithm – Secret Key

- **Symmetric Encryption: Salsa20 stream cipher**
  - **Not AES** ***(Should we care?)***
    - Positive opinion (from DJB, author) https://cr.yp.to/streamciphers/why.html
    - Neutral opinion (from Matthew Green, cryptographer) https://blog.cryptographyengineering.com/2012/10/09/so-you-want-to-use-alternative-cipher/
    - Standardized in European eSTREAM cipher competition
- **Message Authentication: Poly1305 MAC**

Libsodium Languages

- C (Native, API Provided)
- Bindings for other languages
- Python
  - **PyNaCL** - https://github.com/pyca/pynacl
  - LibNaCL - https://github.com/saltstack/libnacl
  - Csodium – Not suggested (limited feature subset)
  - Pysodium – Not suggested (only for Python 2.7)
- .NET, Go, Java, Ruby, Rust, Swift, …
  - https://download.libsodium.org/doc/bindings_for_other_languages/

Installation Instructions

C:

```
$ sudo apt-get install build-essential
$ wget https://download.libsodium.org/libsodium/releases/LATEST.tar.gz
$ tar -xzf LATEST.tar.gz
$ cd libsodium-stable
$ ./configure
$ make && make check
# Should see following printed after test suite runs:  PASS:  70
$ sudo make install
```

Python:

```
$ sudo apt install python3 python3-pip
$ pip3 install --upgrade pip
$ pip3 install pynacl
$ pip3 install --upgrade pynacl  # If needed
```

Lab 12

Your program must transmit to the **`cyberlab.pacific.edu:12001`** server the following payload in a "HTTP-like" format as an ASCII string:

```
CRYPTO 1.0 REQUEST\r\n
Name: Your Name\r\n
PublicKey: <Base16 string of your public key>\r\n
\r\n
```

After transmitting a packet to the server, your program must also receive a TCP reply from the server with the following payload:

```
CRYPTO 1.0 REPLY\r\n
Name: Jeff Shafer\r\n
PublicKey: <Base16 string of Shafer's public key>\r\n
Ciphertext: <Base16 string of ciphertext>\r\n
\r\n
```

World’s laziest Python socket program with absolutely no error checking (exception handling)

```
#!/usr/bin/env python3

import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((hostname, port))
msg = "Unicode string to send"
raw_bytes = bytes(msg, 'ascii’)
s.sendall(raw_bytes)
response = s.recv()
s.close()
```

This is better exception handling…

```
try:
    # Send the string
    s.sendall(raw_bytes)
except socket.error as msg:
    print("Error: sendall() failed")
    print("Description: " + str(msg))
    sys.exit()
```

https://docs.python.org/3/library/socket.html

Base16 Encoding

Binary values → Printable Text
(for transport)

- Input: 4-bit value
- Output: ASCII digits 0-9 and letters A –F
- Can convert in Python using
  - `base64.b16encode()`
  - `base64.16decode()`

https://docs.python.org/3/library/base64.html