



An Overview on Encryption in C++

Jens Weller

CppCon 2015

NDC Oslo 2015
C++Now 2015

About me



- C++ Evangelist
 - @meetingcpp
- C++ since '98
- '02-'07 Vodafone
- '07 selfemployed / freelancer in C++
- '12 Meeting C++

Disclaimer

- I'm not an expert in encryption
- This is an overview covering
 - Crpyto++
 - Botan
 - libSodium

Why?

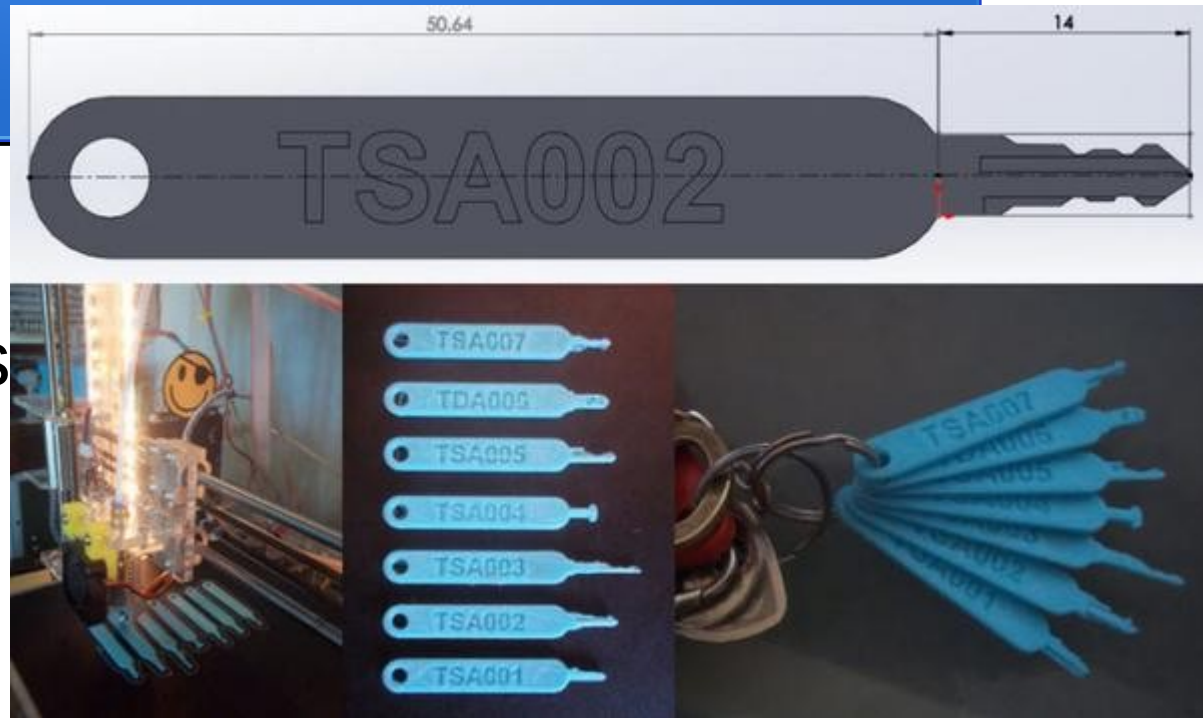
- I was looking for a solution last fall
- What I found
 - Few specialized libraries
 - Lots of documentation
 - Easy to do or understand things wrong
- Gaining more insight in options for encryption
 - Why not give a talk about it?



This talk is about USING encryption.
Not IMPLEMENTING it.

Politics

- Discussion
 - Backdoors
 - If you have
- TSA Keys
 - Backdoor
 - TSA doesn't care they leaked
- Canaries
 - Communicate that you have been gagged

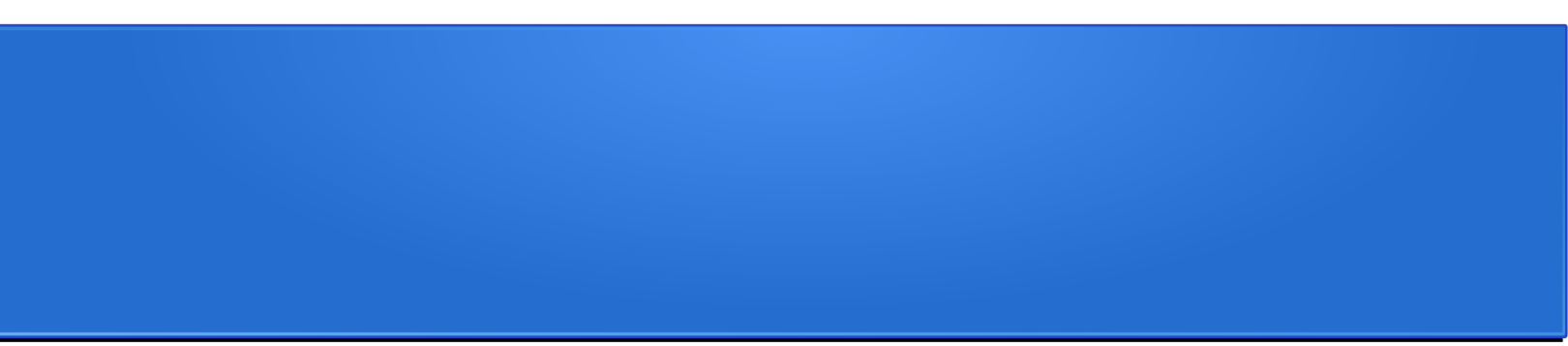


ear

You and your Data

- Are a Target
 - Services
 - Botnets
- Interesting data
 - Email
 - Passwords
 - Logins
- Hardware Resources
- Security Aspects of feature rich, connected embedded devices

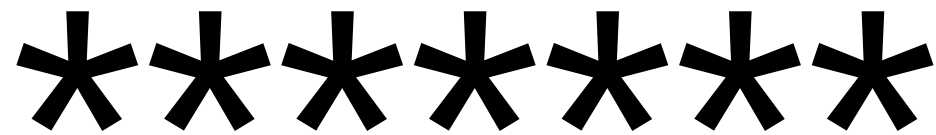




Your attacker lives in the future, but you write your code today.

Short Encryption Update

- Symmetric
 - One Key
 - AES
- A-Symmetric
 - Two Keys
 - Public
 - Private
 - RSA
- Cipher modes
 - Block cipher
 - Stream cipher
- HTTPS
 - Transport Encryption
 - Not for UDP
 - No Data Encryption



Storing Passwords

- Authentication
 - Never as plaintext
 - `hash(pw + salt)`
- Logins
 - Plaintext?
 - Encrypt?

Encrypting Passwords

- Symmetric:
 - Where to store the key?
- A-Symmetric
 - Private key
 - Encrypted with password...
- Plaintext
 - Mutate to real pw?
- OS API?
- I'm still thinking on that problem...



Encryption and C++

Lingua Franca

- unsigned char
 - Most interfaces build up on this
 - `std::string` etc.
 - `char`

C++ Standard & Encryption

- `<random>?`
 - Nope
 - `random_device`
 - Not guaranteed
 - `/dev/urandom`
 - `libc++`
 - `libstdc++`
- Same for boost random

C++ Encryption Libraries

- Cryptopp
- Botan
- libSodium
 - Fork of libNaCl
- OpenSSL
 - libCrypto
- QCA
 - Encryption based on Qt4

C++ Encryption Libraries

- Cryptopp
 - 5.6.2
- Botan
- libSodium
 - Fork of libNaCl
- C++03
- C++11
 - 5.6.3
- Boost License
- C++03(stable)
- C++11(dev)
 - Use this one
- BSD-2
- C
- ISC License

Crypto Examples

- AES
 - Cryptopp
- RSA
 - Botan
- Cryptobox
 - libSodium

AES

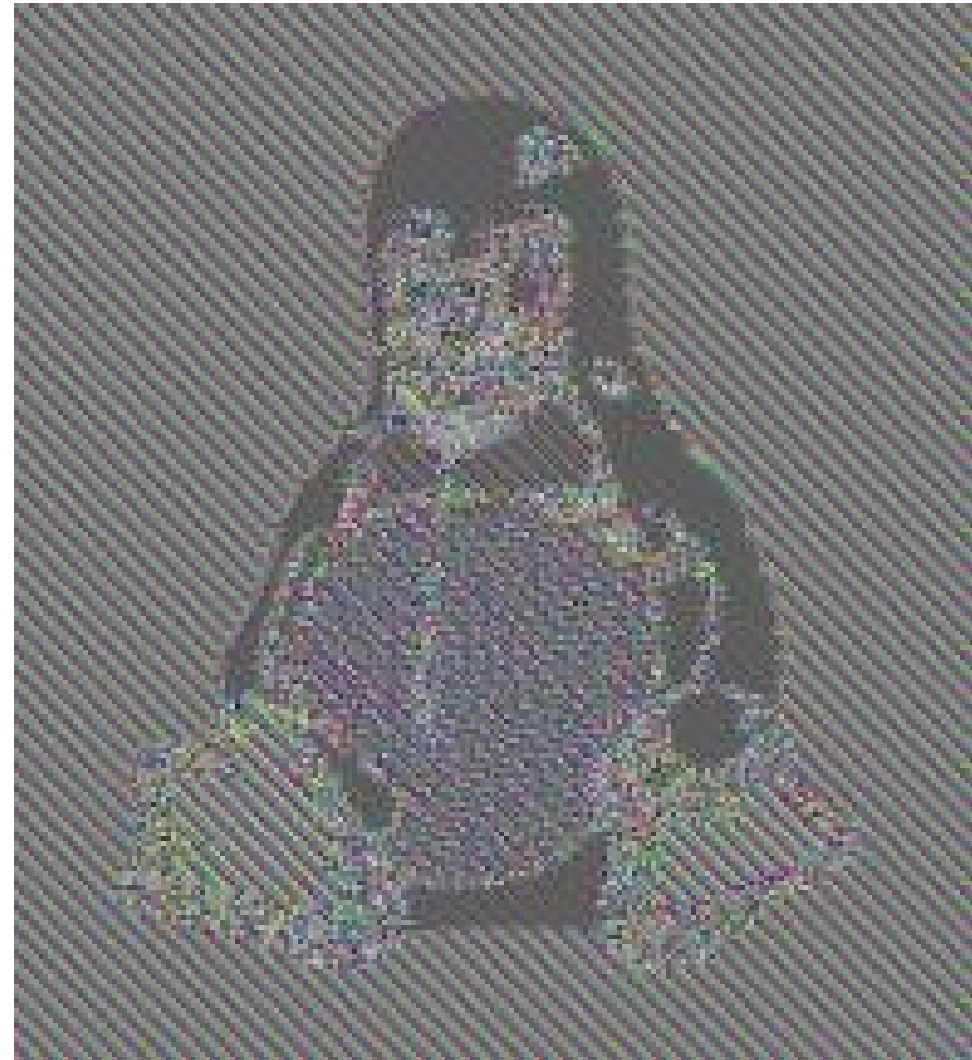
- Advanced Encryption Standard
 - Block Cipher
 - Symmetric
 - Widely used
 - Operates on modes
 - Needs to be initialized

AES - Modes

- AES has different modes
 - ECB – Electronic Code Book
 - CBC – Cipher Block Chaining
 - OFB – Output FeedBack
 - CFB – Cipher FeedBack
 - CTR – Counter Mode
 - Newer modes:
 - EAX & GCM

ECB

- Not really



Jens Weller – Meeting C++
Image: Larry Ewing, lewing@isc.tamu.edu, The GIMP

CBC

- Cipher Block Chaining

- +
- Secure
 - When used properly
- Parallel decryption

- -
- No Parallel encryption
- Known attacks
 - Malleability
 - Secure, when done correctly

OFB

- “Stream Cipher Mode”

- +
 - Key stream
 - Computable in advance
 - Fast hardware implementation
- -
 - Security model is questionable
 - Misconfiguration can lead to short key stream cycles

CFB

- Cipher Feedback
- “CBC backwards”
- +
- Small footprint
- Parallel decryption
- -
- Not very common

CTR

- Counter Mode
- +
- Secure
 - When done right
- Parallel en/decryption
- -
- ?

But wait! There is more!

- EAX and GCM
 - Authentication
 - Encryption
 - Based on CTR
- Which to choose?
 - Depends
 - EAX & GCM
 - CTR
 - CBC/CFB

IV – Initialization Vector

- unsigned char[16];
- Must be random
 - Not pseudo random
- Can be public!
- Do not reuse!
- Usually Libraries provide facilities for generating random bytes.

AES - Cryptopp

Code example

AES – Keys (Cryptopp)

- 16 / 32 bytes

```
AutoSeededRandomPool rnd;
```

```
// Generate a random key
```

```
SecByteBlock key(0x00, AES::DEFAULT_KEYLENGTH);
```

```
rnd.GenerateBlock( key, key.size() );
```

```
// Generate a random IV
```

```
byte iv[AES::BLOCKSIZE];
```

```
rnd.GenerateBlock(iv, AES::BLOCKSIZE);
```

AES – Keys (Cryptopp)

- 16 / 32 bytes

```
AutoSeededRandomPool rnd;
```

```
// Generate a random key
```

```
SecByteBlock key(0x00, AES::DEFAULT_KEYLENGTH);
```

```
rnd.GenerateBlock( key, key.size() );
```

```
// Generate a random IV
```

```
byte iv[AES::BLOCKSIZE];
```

```
rnd.GenerateBlock(iv, AES::BLOCKSIZE);
```

AES – Keys (Cryptopp)

- 16 / 32 bytes

```
AutoSeededRandomPool rnd;
```

```
// Generate a random key
```

```
SecByteBlock key(0x00, AES::DEFAULT_KEYLENGTH);
```

```
rnd.GenerateBlock( key, key.size() );
```

```
// Generate a random IV
```

```
byte iv[AES::BLOCKSIZE];
```

```
rnd.GenerateBlock(iv, AES::BLOCKSIZE);
```

AES – Keys (Cryptopp)

- 16 / 32 bytes

```
AutoSeededRandomPool rnd;
```

```
// Generate a random key
```

```
SecByteBlock key(0x00, AES::DEFAULT_KEYLENGTH);
```

```
rnd.GenerateBlock( key, key.size() );
```

```
// Generate a random IV
```

```
byte iv[AES::BLOCKSIZE];
```

```
rnd.GenerateBlock(iv, AES::BLOCKSIZE);
```


AES Encryption

```
char plainText[] = "Hello! How are you.";
int messageLen = (int)strlen(plainText) + 1;
// Encrypt
CFB_Mode<AES>::Encryption cfbEncryption(key, key.size(), iv);
cfbEncryption.ProcessData(
(byte*)plainText,
(byte*)plainText,
messageLen);
```

AES Encryption

```
char plainText[] = "Hello! How are you.";
int messageLen = (int)strlen(plainText) + 1;

// Encrypt
CFB_Mode<AES>::Encryption cfbEncryption(key, key.size(), iv);
cfbEncryption.ProcessData(
(byte*)plainText,
(byte*)plainText,
messageLen);
```

AES Encryption

```
char plainText[] = "Hello! How are you.";
int messageLen = (int)strlen(plainText) + 1;

// Encrypt
CFB_Mode<AES>::Encryption cfbEncryption(key, key.size(), iv);
cfbEncryption.ProcessData(
(byte*)plainText,
(byte*)plainText,
messageLen);
```

AES - Decryption

```
// Decrypt
```

```
typedef CFB_Mode<AES> mode;
```

```
mode::Decryption cfbDecryption(key, key.size(), iv);
```

```
cfbDecryption.ProcessData(
```

```
    (byte*)plainText,
```

```
    (byte*)plainText,
```

```
    messageLen);
```

AES

- Cryptopp
 - Use a random iv & key
 - The key is not a password
- Modes
 - Select the right one for your use case
- Padding
 - Offset before the encryption
 - Cryptopp examples do not use it

RSA

- Asymmetric Cipher
- 2 Keys
 - Public
 - Private
- Public Key
 - Encryption
 - Can be shared
- Private Key
 - Must not be shared
 - Should be protected

Botan

- namespace Botan
- Botan initialization
 - Botan::LibraryInitializer
 - Create an instance (stack!)
 - Can throw (try & catch)
- A collection of encryption algorithms

RSA - Botan

- Where do you get a private Key from?
 - `RSA_PrivateKey`
 - `(RandomNumberGenerator& rng, size_t bits)`

RSA - Botan

Code example

RSA encryption with botan

```
std::string text = "abc";  
AutoSeeded_RNG rng;  
RSA_PrivateKey key(rng, 1024);  
std::string pub = X509::PEM_encode(key);  
std::string priv = PKCS8::PEM_encode(key);  
DataSource_Memory key_pub(pub);  
DataSource_Memory key_priv(priv);
```

RSA encryption with botan

```
std::string text = "abc";
```

```
AutoSeeded_RNG rng;
```

```
RSA_PrivateKey key(rng, 1024);
```

```
std::string pub = X509::PEM_encode(key);
```

```
std::string priv = PKCS8::PEM_encode(key);
```

```
DataSource_Memory key_pub(pub);
```

```
DataSource_Memory key_priv(priv);
```

RSA encryption with botan

```
std::string text = "abc";  
AutoSeeded_RNG rng;  
RSA_PrivateKey key(rng, 1024);  
std::string pub = X509::PEM_encode(key);  
std::string priv = PKCS8::PEM_encode(key);  
DataSource_Memory key_pub(pub);  
DataSource_Memory key_priv(priv);
```

RSA encryption with botan

```
std::string text = "abc";  
AutoSeeded_RNG rng;  
RSA_PrivateKey key(rng, 1024);  
std::string pub = X509::PEM_encode(key);  
std::string priv = PKCS8::PEM_encode(key);  
DataSource_Memory key_pub(pub);  
DataSource_Memory key_priv(priv);
```

RSA Keysetup

```
X509_PublicKey *pub_rsa = X509::load_key(key_pub);  
PKCS8_PrivateKey *priv_rsa = PKCS8::load_key(key_priv, rng);  
  
auto *enckey = dynamic_cast<PK_Encrypting_Key*>(pub_rsa);  
auto *deckey = dynamic_cast<PK_Decrypting_Key*>(priv_rsa);  
  
PK_Ecryptor *enc = get_pk_encryptor(*enckey, "EME1(SHA-256)");  
PK_Decryptor *dec = get_pk_decryptor(*deckey, "EME1(SHA-256)");
```

RSA Keysetup

```
X509_PublicKey *pub_rsa = X509::load_key(key_pub);  
PKCS8_PrivateKey *priv_rsa = PKCS8::load_key(key_priv, rng);  
  
auto *enckey = dynamic_cast<PK_Encrypting_Key*>(pub_rsa);  
auto *deckey = dynamic_cast<PK_Decrypting_Key*>(priv_rsa);  
  
PK_Encoder *enc = get_pk_encoder(*enckey, "EME1(SHA-256)");  
PK_Decryptor *dec = get_pk_decryptor(*deckey, "EME1(SHA-256)");
```

RSA Keysetup

```
X509_PublicKey *pub_rsa = X509::load_key(key_pub);  
PKCS8_PrivateKey *priv_rsa = PKCS8::load_key(key_priv, rng);  
  
auto *enckey = dynamic_cast<PK_Encrypting_Key*>(pub_rsa);  
auto *deckey = dynamic_cast<PK_Decrypting_Key*>(priv_rsa);  
  
PK_Encoder *enc = get_pk_encoder(*enckey, "EME1(SHA-256)");  
PK_Decryptor *dec = get_pk_decryptor(*deckey, "EME1(SHA-256)");
```


RSA Keysetup

```
X509_PublicKey *pub_rsa = X509::load_key(key_pub);  
PKCS8_PrivateKey *priv_rsa = PKCS8::load_key(key_priv, rng);  
  
auto *enckey = dynamic_cast<PK_Encrypting_Key*>(pub_rsa);  
auto *deckey = dynamic_cast<PK_Decrypting_Key*>(priv_rsa);  
  
PK_Ecryptor *enc = get_pk_encryptor(*enckey, "EME1(SHA-256)");  
PK_Decryptor *dec = get_pk_decryptor(*deckey, "EME1(SHA-256)");
```

RSA En/Decrypting

```
byte msg[text.size()];
```

```
SecureVector<byte> ciphertext  
    = enc->encrypt(msg, sizeof(msg), rng);
```

```
SecureVector<byte> plaintext  
    = dec->decrypt(ciphertext, ciphertext.size());
```

Cryptobox

- AES & RSA
 - Can be tricky to setup
- What if just a good encryption is needed?
- A Cryptobox
 - Hides implementation details
 - Exposes an easier interface for encryption

Cryptobox approach

//pseudo code

namespace cryptobox

{

Buffer decrypt(key,buffer,algo);

Buffer encrypt(key,buffer,algo);

}

Cryptobox

- Botan
 - Based on Serpent (Block cipher)
- Libsodium
 - Different cryptoboxes
 - Symmetric
 - A-Symmetric

libSodium

- Fork of libNaCl
 - Goal is to make encryption accessible
- C library
 - C++ wrappers exist
- Initialization:
 - `sodium_init()`

Cryptobox - libSodium

Code example

libSodium

```
#define MESSAGE ((const unsigned char *) "test")  
#define MESSAGE_LEN 4  
#define CIPHERTEXT_LEN  
(crypto_secretbox_MACBYTES + MESSAGE_LEN)  
  
unsigned char nonce[crypto_secretbox_NONCEBYTES];  
unsigned char key[crypto_secretbox_KEYBYTES];  
unsigned char ciphertext[CIPHERTEXT_LEN];
```


libSodium

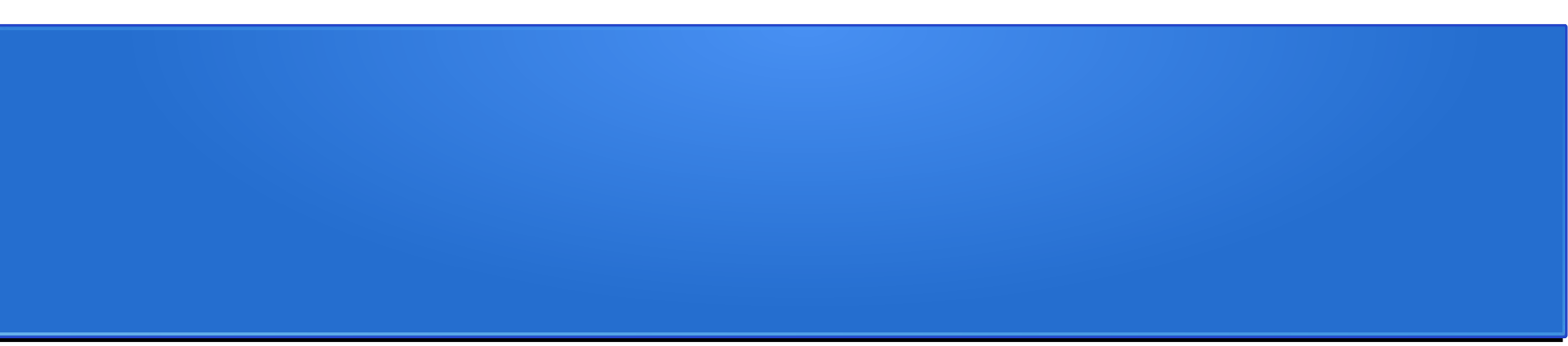
```
#define MESSAGE ((const unsigned char *) "test")  
#define MESSAGE_LEN 4  
#define CIPHERTEXT_LEN  
(crypto_secretbox_MACBYTES + MESSAGE_LEN)
```

```
unsigned char nonce[crypto_secretbox_NONCEBYTES];  
unsigned char key[crypto_secretbox_KEYBYTES];  
unsigned char ciphertext[CIPHERTEXT_LEN];
```

libSodium

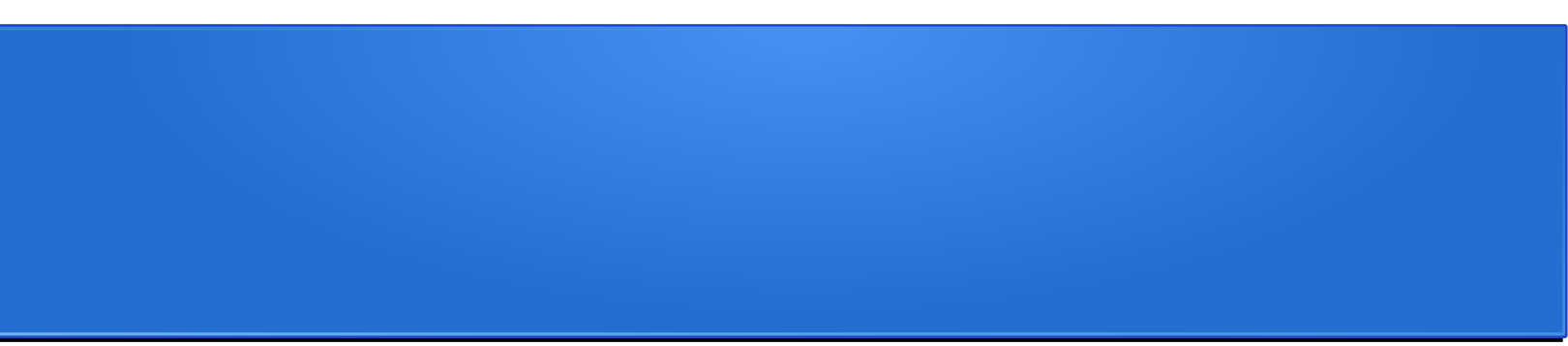
```
#define MESSAGE ((const unsigned char *) "test")  
#define MESSAGE_LEN 4  
#define CIPHERTEXT_LEN  
(crypto_secretbox_MACBYTES + MESSAGE_LEN)
```

```
unsigned char nonce[crypto_secretbox_NONCEBYTES];  
unsigned char key[crypto_secretbox_KEYBYTES];  
unsigned char ciphertext[CIPHERTEXT_LEN];
```



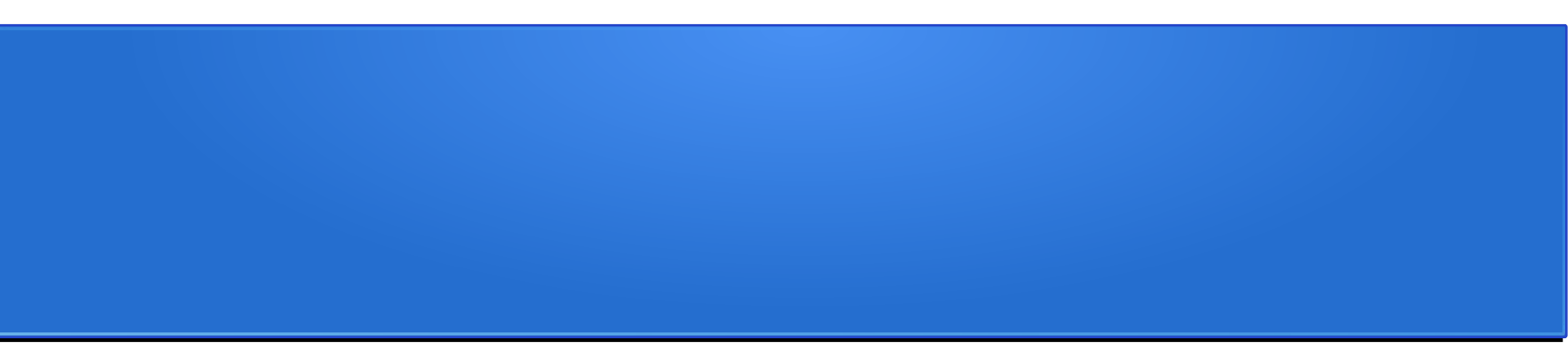
```
randombytes_buf(nonce, sizeof nonce);
randombytes_buf(key, sizeof key);
crypto_secretbox_easy(ciphertext, MESSAGE, MESSAGE_LEN,
nonce, key);

unsigned char decrypted[MESSAGE_LEN];
if (crypto_secretbox_open_easy(decrypted, ciphertext,
CIPHERTEXT_LEN, nonce, key) != 0) {
    /* message forged! */
}
```



```
randombytes_buf(nonce, sizeof nonce);  
randombytes_buf(key, sizeof key);  
crypto_secretbox_easy(ciphertext, MESSAGE, MESSAGE_LEN,  
nonce, key);
```

```
unsigned char decrypted[MESSAGE_LEN];  
if (crypto_secretbox_open_easy(decrypted, ciphertext,  
CIPHERTEXT_LEN, nonce, key) != 0) {  
    /* message forged! */  
}
```



```
randombytes_buf(nonce, sizeof nonce);  
randombytes_buf(key, sizeof key);  
crypto_secretbox_easy(ciphertext, MESSAGE, MESSAGE_LEN,  
nonce, key);
```

```
unsigned char decrypted[MESSAGE_LEN];  
if (crypto_secretbox_open_easy(decrypted, ciphertext,  
CIPHERTEXT_LEN, nonce, key) != 0) {  
    /* message forged! */  
}
```

In the box...

- Symmetric
 - Encryption: XSalsa20 stream cipher
 - Authentication: Poly1305 MAC
- A-Symmetric
 - Key exchange: Curve25519
 - Encryption: XSalsa20 stream cipher
 - Authentication: Poly1305 MAC

Final thoughts

- Encrypt critical data
- Botan and Crypto++
 - Are a collection of encryption algorithms
- A Cryptobox
 - easy and save encryption
 - libSodium – symmetric & a-symmetric
 - Botan - symmetric

The End

Questions?