Problems and pitfalls in the implementation of cryptographic systems

DeCifris Mediolanensibus, October 12, 2021

Enrico Zimuel - *Principal Software Engineer* at Elastic

Agenda

- Introduction to cryptography engineering
- Implementation errors and secure software
- Keeping secrets
- Implementation issues: randomness, seed, user’s password, storing password
- Side-channel attack, Timing attack
- Crypto libraries: NaCl, libsodium
- Examples: use libsodium in PHP

Cryptography engineering

- *“Cryptographic engineering is the name we have coined to refer to the theory and practice of engineering of cryptographic systems”*
  Çetin Kaya Koç in Cryptographic Engineering
- A **cryptographic engineer** designs, implements, tests, and validates cryptographic systems
- She is also interested in **breaking** them for the purpose of checking their robustness and also building countermeasures to prevent or mitigate future attacks

Weakest link property

A security system is only as strong as its weakest link

Adversarial setting

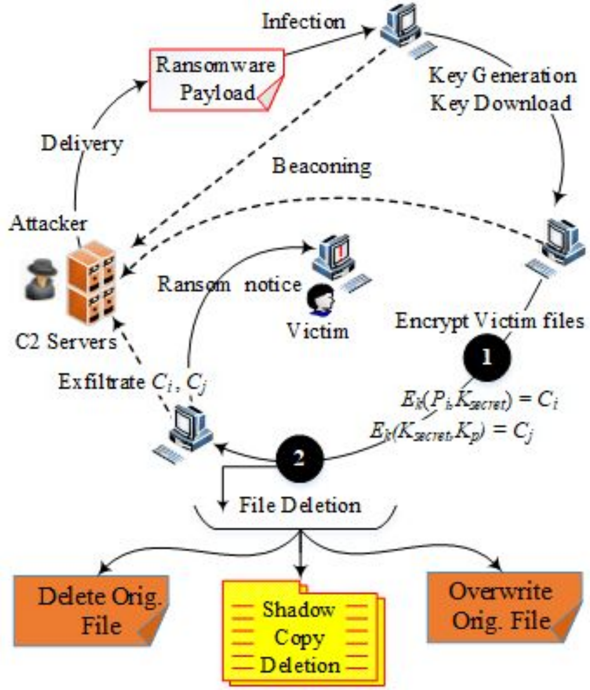
- One of the biggest differences between security systems and almost any other type of engineering is the **adversarial setting**
- Most engineers have to contend with problems like heat, cold, humidity, pressure, etc.
- All these factors affect designs, but their effect is **fairly predictable**
- In security an opponent is intelligent, clever, malicious
- They don’t play by the rules, and they are **completely unpredictable**

Cryptographers are professional paranoids

Source: John Klossner technology cartoons

Threat model

- **Every system can be attacked**
- The whole point of a security system is to provide access to some people and not to others
- In the end, you will always have to trust some people in some way, and these people in turn can attack your system
- **What are you trying to protect against?**



Cryptography is not the solution

- **Cryptography is not the solution to your security problems**
- It might be part of the solution, or it might be part of the problem
- In many situations, cryptography starts out by making the problem worse, and it isn't at all clear that using cryptography is an improvement
- **Cryptography has many uses**
- It is a crucial part of many good security system
- It can also make systems weaker when used in inappropriate ways
- It's very dangerous, it can provide a **feeling of security** but not actual security

Cryptography is very difficult

- **Cryptography is fiendishly difficult**
- Even seasoned experts design systems that are broken a few years later
- This is common enough that we are not surprised when it happens
- The weakest-link property and the adversarial setting conspire to make life for a cryptographer very hard
- Another significant problem is the **lack of testing**
- There is no known way of testing whether a system is secure
- A bad cryptography looks just like good cryptography, until it is seriously attacked

Cryptography is the easy part

- Even though cryptography itself is difficult, it is still one of the easy parts of a security system
- A cryptographic component has fairly well defined boundaries and requirements
- An entire security system is much more difficult to clearly define, since it involves many more aspects
- Another huge problem is the software quality, security cannot be effective if a software contains thousands of bug that lead to security holes

Implementation errors

- **Implementation errors are by far the biggest security problem in real-world systems**
- One major part is, as always, the operating system (OS)
- But none of the operating system in widespread use is designed with security as a primary goal
- The logical conclusion is that **is impossible to implement a security system**
- When we design a cryptographic system, we do our best to make sure that at least our part is secure

Secure software

- **We can write correct software**
- This is **not good enough for a security system**
- Correct software has a specified functionality, eg. if you click a button A then B will happen
- Secure software has an additional requirement: **a lack of functionality**; eg. no matter what an attacker does, she cannot do X
- This is very different, you can test for functionality, but not for lack of functionality
- **We actually don’t know how to create secure code**!

Keeping secrets

Keeping secrets

- **Anytime you work with cryptography, you are dealing with secrets**
- For the secure channel we have two type of secrets:
    - **keys**;
    - **data**;
- Both of these are transient secrets, we don't have to store them for a long time
- The data is only stored while we process each message
- The key are only stored for the duration of the secure channel
- **Transient secrets are keep in memory**

Wiping state

- A basic rule of writing security software is to wipe any information as soon as you no longer need it
- The longer you keep it, the higher the chance that someone will be able to access it
- Free a variable (deallocates the memory) is not enough, you need to override the old data
- This is related to the programming language used
  - in C using memset();
  - in C++ using destructor;
  - more difficult in Java because the heap is garbage-collected

Wiping state in Java

- One solution to mitigate the heap state is to run a program with **try-finally** statement
- At least we can ensure that the finalization routines are run at program exit
- The finally block contains some code to force a garbage collect, using **System.gc()** and **System.runFinalization()**

Swap file

- Most operating systems use a virtual memory system to increase the number of programs that can be run in parallel
- While a program is running, not all of its data is kept in memory; some is stored in a **swap file**:
  - when the program tries to access data that is not in memory, the program is interrupted;
  - the virtual memory system reads the required data from the swap file into a piece of memory;
  - the program is allowed to continue;
  - when the virtual memory system requires more free memory, it will take an arbitrary piece of memory and write it to the swap file

Swap file (2)

- In some operating systems there are system calls that you can use to inform the virtual memory system that specific parts of the memory are not to be swapped out:
  - in Windows, we can use the **VirtualLock()** API;
  - in Unix systems, the **mlock()** system call is often implemented in such a way that locked memory is never swapped to disk

Cache

- Modern computers have hierarchy of memories
- The cache keeps a copy of the most recently used data from the main memory
- This is a smaller but faster memory
- It is not a great danger from a security perspective since only the OS can access the cache memory
- We need to trust the OS, there is very little we can do about this

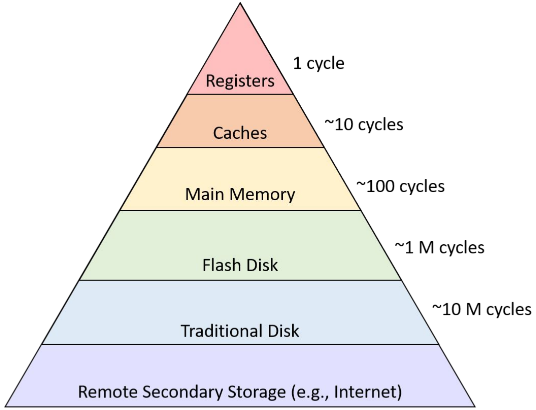


Image source: Dive Into Systems

Data retention by memory

- Simply overwriting data in memory does not delete the data
- This effect depends on the type of memory involved, basically when if you store data in a memory location, the location slowly starts to “learn” the data
- If you switch off the computer, the old data cannot be completely lost
- It is arguable whether this is a significant threat
- Workaround: instead of storing $\boldsymbol{m}$, we generate a random string $\boldsymbol{R}$ and we store $\boldsymbol{R}$ and $\boldsymbol{R} \oplus \boldsymbol{m}$, see [15] and [16] in references
- Preventing data retention: Eraser (Windows), shred (Linux)

Data integrity

- In addition to keeping secrets, we should protect the integrity of the data we are storing
- We use MAC to protect the integrity during the transit but if the data can be modified in memory we still have problems

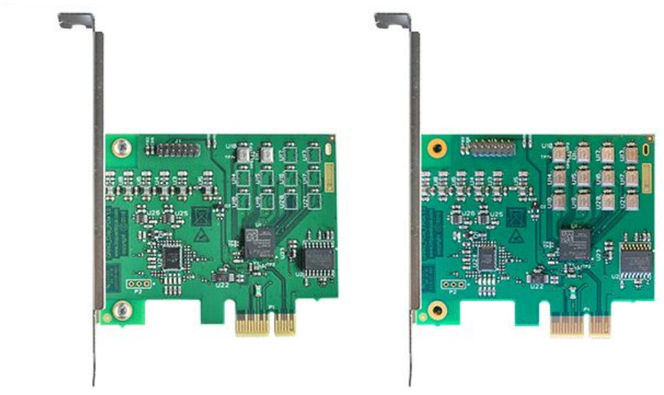
Implementation issues

Randomness

- Generating good randomness is a vital part of many cryptographic operations and it is one of the most difficult ones
- In computer languages we use **pseudorandom data** (not really random)
- It is generated from a **seed** by a **deterministic algorithm**
- If you know the seed you can predict the pseudorandom data
- In cryptography we use pseudorandom number generator (**PRNG**) that are **considered strong**: even if an attacker sees a lot of the random data generated by PRNG, she should not be able to predict anything about the rest of the output

Seed

- The seed is a crucial part of a PRNG
- How can we choose a random seed?
  - Windows: Cryptography API, Next Generation
  - Linux: getrandom()
  - Linux: /dev/urandom
  - Quantum Random Number Generation



Source image: Quantis QRNG PCIe New Generation

User’s password

- **Not random**
- Predictable (most of the time)
- Only a subset of ASCII codes (typically 68 vs 256)
- **Never use it as encryption/authentication key!**
- Use Key Derivation Function (KDF) to generate a key from a user’s password
- Eg. PBKDF2, Argon2i, Lyra2, Catena, yescrypt, Makwa, Balloon hashing

How to store user’s password

- Hashing is the best approach to store a user’s password (eg. in a file or a database)
- Which hash algorithm to use?
- **MD5** and **SHA** family hash are not good, they are vulnerable to brute force attack using GPU (few seconds in some cases)
- Good hash algorithms are the following adaptive functions:
  - **bcrypt** (CPU intensive)
  - **scrypt** (CPU and memory intensive)
  - **Argon2** (CPU, memory and degree of parallelism intensive)

Bruteforce attack

- A graphics processing unit (**GPU**) is a specialized CPU used in video games to execute multiple operation in parallel
- It can be used to run hash algorithms in parallel to perform a brute force attack
- A GPU has **thousands of core** (eg. 4000), a CPU just multiple (eg. 16)
- Using hashcat software and GPUs you can crack a 8 characters password in seconds!

Source image: 25-GPU cluster

Side-channel attacks

Side-channel attack

- Attack based on **information gained from the implementation** of a computer system, rather than weaknesses in the implemented algorithm itself

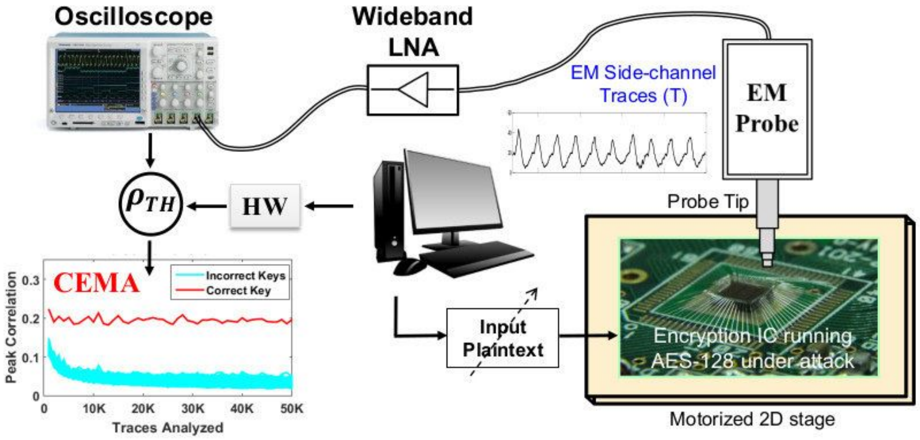
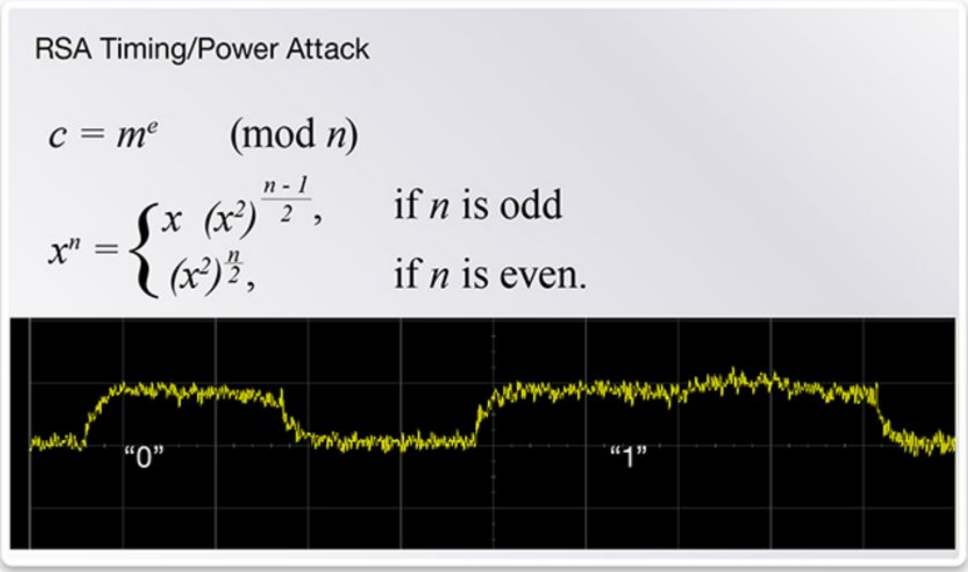


Image source: STELLAR, a Generic EM Side-Channel Attack Protection through Ground-Up Root-cause Analysis

Decode RSA key using power analysis



Source: [Protecting Against Side-Channel Attacks with an Ultra-Low Power Processor]

Timing attack

- An attacker **measures the CPU time** to perform some procedures involving a secret (e.g. encryption key). If this time depends on the secret, the attacker may be able to deduce information about the secret
- In 2006 *A. Shamir, E.Tromer and D.A. Osvik* used a timing attack to extract the full encryption key in **65 ms** using a Linux dm-crypt device with 128-bit AES in ECB mode (see [18] in references)

Prevent timing attack

- We need to reduce the information that an attacker can retrieve measuring the execution time
- For instance:
  - implement algorithm with constant execution time, eg. not related to the size of the key
  - avoid the usage of lookup tables in encryption algorithms to prevent cache timing effects

Example: compare strings

- What information an attacker can deduce from the following code?

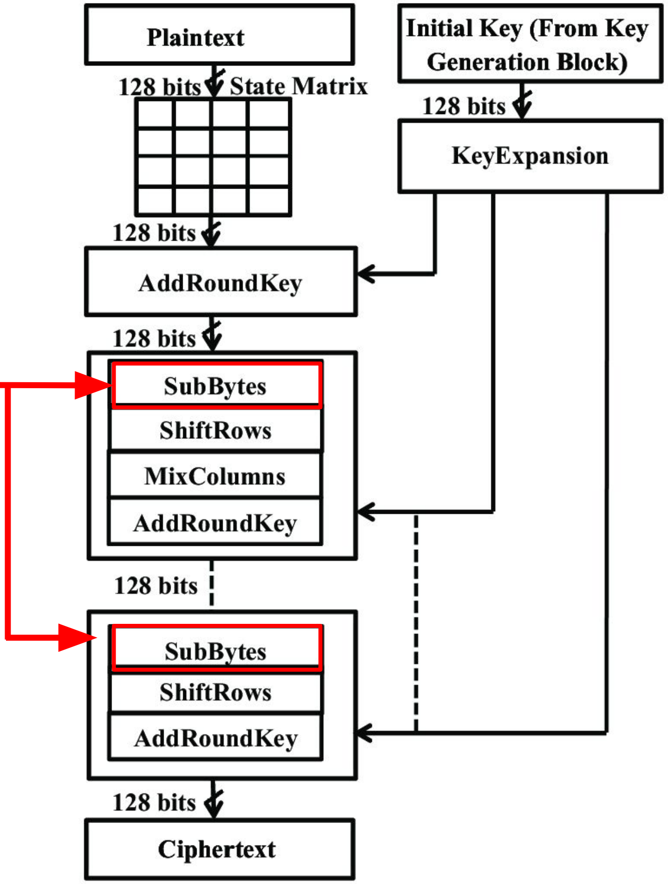
```
function compare(string $expected, string $actual): bool
{
    $lenExpected = strlen($expected);
    $lenActual   = strlen($actual);
    if ($lenExpected !== $lenActual) {
        return false;
    }
    for($i=0; $i < $lenActual; $i++) {
        if ($expected[$i] !== $actual[$i]) {
            return false;
        }
    }
    return true;
}
```

Cache-timing attacks

- **Cache-timing attacks** are software side-channel attacks exploiting the timing variability of data loads from memory
- This variability is due to the fact that all modern microprocessors use a hierarchy of caches to reduce load latency
- If a load operation can retrieve data from one of the caches (**cache hit**), the load takes less time than if the data has to be retrieved from RAM (**cache miss**)

S-box in AES

- The S-box maps an 8-bit input, c, to an 8-bit output
- The S-box is used in **SubBytes** function
- C. Ashokkumar et. al. showed that **“S-Box” Implementation of AES is NOT side channel resistant**, using a lookup table (see [21] in references)



S-box in C

https://github.com/kokke/tiny-AES-c/blob/master/aes.c#L76-L96

```
// The lookup-tables are marked const so they can be placed in read-only storage instead of RAM
// The numbers below can be computed dynamically trading ROM for RAM -
// This can be useful in (embedded) bootloader applications, where ROM is often limited.
static const uint8_t sbox[256] = {
  //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
  0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
  0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
  0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
  0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
  0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
  0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
  0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
  0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
  0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
  0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
  0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
  0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
  0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
  0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
  0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
  0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};
```

Bitslicing

- The **bitslicing** technique has been introduced by Eli Biham in 1997 (see [20] in references)
- Essentially, bitslicing simulates a hardware implementation in software: the entire algorithm is represented as a sequence of atomic Boolean operations
- **This sequence is executed in constant time**
- We can use it to implement S-box in AES (see [19] in references)

Bitslicing in AES

tiny-AES-c, implementation

```
static void SubBytes(state_t* state)
{
 uint8_t i, j;
 for (i = 0; i < 4; ++i)
 {
   for (j = 0; j < 4; ++j)
   {
     (*state)[j][i] =
getSBoxValue((*state)[j][i]);
   }
 }
}
```

https://github.com/kokke/tiny-AES-c/blob/master/aes.c#L251-L261

AES bitslicing

```
static void SubBytes(state_t* state)
{
 AES_state s = {{0}};
 LoadBytes(&s, state);
 SBoxBS(&s);
 SaveBytes(state, &s);
}
```

```
static void SBoxBS(AES_state *s) {
   uint16_t U0 = s->slice[7], U1 = s->slice[6], U2 = s->slice[5], U3 = s->slice[4];
   uint16_t U4 = s->slice[3], U5 = s->slice[2], U6 = s->slice[1], U7 = s->slice[0];

   uint16_t T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16;
   uint16_t T17, T18, T19, T20, T21, T22, T23, T24, T25, T26, T27, D;
   uint16_t M1, M6, M11, M13, M15, M20, M21, M22, M23, M25, M37, M38, M39, M40;
   uint16_t M41, M42, M43, M44, M45, M46, M47, M48, M49, M50, M51, M52, M53, M54;
   uint16_t M55, M56, M57, M58, M59, M60, M61, M62, M63;

   T1 = U0 ^ U3;
   T2 = U0 ^ U5;
   T3 = U0 ^ U6;
   ...
}
```

Crypto library: NaCl

NaCl

- NaCl: Networking and Cryptography library
- High-speed software library for network communication, encryption, decryption, signatures, etc
- Developed by Prof. Daniel J. Bernstein, and others
- Highly-secure primitives and constructions, implemented with extreme care to avoid **side-channel attacks**

Sodium

- **Sodium** (libsodium) is a fork of NaCl
- A portable, cross-compilable, installable, packageable, API-compatible version of NaCl
- Same implementations of crypto primitives as NaCl
- Shared library and a standard set of headers (portable implementation)
- Official web site: libsodium.org

Sodium: features

- Authenticated public-key and authenticated shared-key encryption
- Public-key and shared-key signatures
- Hashing
- Keyed hashes for short messages
- Secure pseudo-random numbers generation
- Zeroing memory

Sodium: algorithms

- **Curve25519**, Diffie–Hellman key-exchange function
- **Salsa20**, ChaCha20 stream ciphers
- **Poly1305**, message-authentication code
- **Ed25519,** public-key signature system
- **Argon2** and **Scrypt,** password hashing
- **AES-GCM**, authenticated encryption algorithm

Examples: use libsodium in PHP

Sodium in PHP

- Available (as standard library) from **PHP 7.2**
- 85 functions with prefix **sodium_**
- e.g. sodium_crypto_box_keypair()

Symmetric encryption

```php
// code1.php at https://github.com/ezimuel/sodium-php-talk
$msg = 'This is a super secret message!';
// Generating an encryption key and a nonce
$key   = random_bytes(SODIUM_CRYPTO_SECRETBOX_KEYBYTES);  // 256 bit
$nonce = random_bytes(SODIUM_CRYPTO_SECRETBOX_NONCEBYTES);  // 24 bytes
// Encrypt
$ciphertext = sodium_crypto_secretbox($msg, $nonce, $key);
// Decrypt
$plaintext = sodium_crypto_secretbox_open($ciphertext, $nonce, $key);

echo $plaintext === $msg ? 'Success' : 'Error';
```

Algorithms: **XSalsa20** for encrypt and **Poly1305** for MAC

Symmetric authentication

```php
// code2.php at https://github.com/ezimuel/sodium-php-talk
$msg = 'This is the message to authenticate!' ;
$key = random_bytes(SODIUM_CRYPTO_SECRETBOX_KEYBYTES);  // 256 bit

// Generate the Message Authentication Code
$mac = sodium_crypto_auth($msg, $key);

// Altering $mac or $msg, verification will fail
echo sodium_crypto_auth_verify($mac, $msg, $key) ? 'Success' : 'Error';
```

Algorithm: **HMAC-SHA512**

Public key encryption

```php
// code3.php at https://github.com/ezimuel/sodium-php-talk
$aliceKeypair = sodium_crypto_box_keypair();
$alicePublicKey = sodium_crypto_box_publickey($aliceKeypair);
$aliceSecretKey = sodium_crypto_box_secretkey($aliceKeypair);

$bobKeypair = sodium_crypto_box_keypair();
$bobPublicKey = sodium_crypto_box_publickey($bobKeypair); // 32 bytes
$bobSecretKey = sodium_crypto_box_secretkey($bobKeypair); // 32 bytes

$msg = 'Hi Bob, this is Alice!';
$nonce = random_bytes(SODIUM_CRYPTO_BOX_NONCEBYTES); // 24 bytes
$keyEncrypt = $aliceSecretKey . $bobPublicKey;
$ciphertext = sodium_crypto_box($msg, $nonce, $keyEncrypt);
$keyDecrypt = $bobSecretKey . $alicePublicKey;
$plaintext = sodium_crypto_box_open($ciphertext, $nonce, $keyDecrypt);
echo $plaintext === $msg ? 'Success' : 'Error';
```

Algorithms:
XSalsa20 for encrypt,
Poly1305 for MAC, and
XS25519 for key exchange

Digital signature

```php
// code4.php at https://github.com/ezimuel/sodium-php-talk
$keypair = sodium_crypto_sign_keypair();
$publicKey = sodium_crypto_sign_publickey($keypair); // 32 bytes
$secretKey = sodium_crypto_sign_secretkey($keypair); // 64 bytes

$msg = 'This message is from Alice';
// Sign a message
$signedMsg = sodium_crypto_sign($msg, $secretKey);
// Or generate only the signature (detached mode)
$signature = sodium_crypto_sign_detached($msg, $secretKey); // 64 bytes
// Verify the signed message
$original = sodium_crypto_sign_open($signedMsg, $publicKey);
echo $original === $msg ? 'Signed msg ok' : 'Error signed msg';
// Verify the signature
echo sodium_crypto_sign_verify_detached($signature, $msg, $publicKey) ?
    'Signature ok' : 'Error signature';
```

Algorithm: **Ed25519**

AEAD AES-256-GCM

```php
// code5.php at https://github.com/ezimuel/sodium-php-talk
$msg = 'Super secret message!';
$key = random_bytes(SODIUM_CRYPTO_AEAD_AES256GCM_KEYBYTES);
$nonce = random_bytes(SODIUM_CRYPTO_AEAD_AES256GCM_NPUBBYTES);
// AEAD encryption
$ad = 'Additional public data';
$ciphertext = sodium_crypto_aead_aes256gcm_encrypt($msg, $ad, $nonce, $key);
// AEAD decryption
$decrypted = sodium_crypto_aead_aes256gcm_decrypt($ciphertext, $ad, $nonce, $key);
if ($decrypted === false) {
   throw new \Exception("Decryption failed");
}
echo $decrypted === $msg ? 'OK' : 'Error';
```

ARGON2i

```
// code6.php at https://github.com/ezimuel/sodium-php-talk
$password = 'password';
$hash = sodium_crypto_pwhash_str (
   $password,
   SODIUM_CRYPTO_PWHASH_OPSLIMIT_INTERACTIVE,
   SODIUM_CRYPTO_PWHASH_MEMLIMIT_INTERACTIVE
); // 97 bytes
echo sodium_crypto_pwhash_str_verify ($hash, $password) ?
    'OK' : 'Error';
```

An example of output:

```
$argon2id$v=19$m=65536,t=2,p=1$EF1BpShRmCYHN7ryxlhtBg$zLZO4IWjx3E...
```

KDF using ARGON2i

```
// code8.php at https://github.com/ezimuel/sodium-php-talk
$password = 'password';
$salt = random_bytes(SODIUM_CRYPTO_PWHASH_SALTBYTES);

$key = sodium_crypto_pwhash(
   32,
   $password,
   $salt,
   SODIUM_CRYPTO_PWHASH_OPSLIMIT_INTERACTIVE,
   SODIUM_CRYPTO_PWHASH_MEMLIMIT_INTERACTIVE
);
```

References

1. N. Ferguson, B. Schneier, Practical Cryptography, John Wiley & Sons, 432 pages, 2003
2. N. Ferguson, B. Schneier, T. Kohno, Cryptography Engineering, 384 pages, 2010
3. Jean-Philippe Aumasson, Serious Cryptography, No Starch Press, 312 pages, 2017
4. Svetlin Nakov, Practical cryptography for developers, SoftUni foundation, 2018
5. Matthew Green, A Few Thoughts on Cryptographic Engineering, blog
6. Çetin Kaya Koç, Cryptographic Engineering, Springer, 522 pages, 2009
7. Nigel P. Smart, Cryptography Made Simple, Springer, 493 pages, 2016
8. Gabriele Paoloni, How to Benchmark Code Execution Times on Intel IA-32 and IA-64, White paper, Intel, 2010
9. Serge Vaudenay, Security Flaws Induced by CBC Padding, Advances in Cryptology — EUROCRYPT 2002. EUROCRYPT 2002. Lecture Notes in Computer Science, vol 2332
10. D. Brumley, D. Boneh, Remote timing attacks are practical, Computer Networks, Volume 48, Issue 5, August 2005, Pages 701-716
11. Daniel Bleichenbacher, Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1, Advances in Cryptology --- CRYPTO '98, Springer
12. C. Rebeiro, D. Mukhopadhyay, S.Bhattacharya, Timing Channels in Cryptography, Springer International Publishing Switzerland, 152 pages, 2015
13. M. Joye, M. Tunstall, Fault Analysis in Cryptography, Springer-Verlag, 356 pages, 2012

References (2)

14. F. R. Henríquez, A. D. Pérez, N. A. Saqib, C. K. Koç, Cryptographic Algorithms on Reconfigurable Hardware, Springer Science+Business Media, 362 pages, 2007
15. Peter Gutmann, Secure Deletion of Data from Magnetic and Solid-State Memory, USENIX Security Symposium Proceedings, 1996
16. G. Di Crescenzo, N. Ferguson, R. Impagliazzo, M. Jakobsson, How To Forget a Secret, STACS 99. STACS 1999. Lecture Notes in Computer Science, vol 1563. Springer
17. Nicolas T. Courtois, All About Side Channel Attacks, Applied Crypto COMPGA12, University College London, 2013
18. D.A. Osvik, A. Shamir, E.Tromer Efficient Cache Attacks on AES, and Countermeasures, Journal of Cryptology, 2009
19. E. Käsper, P. Schwabe, Faster and Timing-Attack Resistant AES-GCM, Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, vol 5747. Springer, 2009
20. Eli Biham, A fast new DES implementation in software , In Fast Software Encryption: 4th InternationalWorkshop, FSE'97, volume 1267 ofLNCS, pages 260–272. Springer, 1997
21. C. Ashokkumar, B. Roy, M. B. S. Venkatesh, B. L. Menezes, "S-Box" Implementation of AES is NOT side channel resistant, Journal of Hardware and System Security, Issue 4, pages 86–97, 2020

Thanks!

Contacts:
enrico (at) zimuel.it
@ezimuel