Licence L2 STS Mention SPI Parcours Informatique
Unité 174EN007
Sécurité Informatique

TD2 - Annexe

```
#ifndef SHA256_H
#define SHA256_H

/****************************** DATA TYPES ******************************/
typedef unsigned char BYTE; // 8-bit byte
typedef unsigned int  WORD; // 32-bit word, change to "long" for 16-bit machines

/*********************** FUNCTION DECLARATIONS **********************/
void sha256(const BYTE data[], size_t len, BYTE hash[]);

#endif   // SHA256_H
```

```
/****************************** HEADER FILES ******************************/
#include <stdlib.h>
#include <stddef.h>
#include <memory.h>
#include "sha256.h"

/****************************** DATA TYPES ******************************/
typedef struct {
        BYTE data[64];
        WORD datalen;
        unsigned long long bitlen;
        WORD state[8];
} SHA256_CTX;

/****************************** MACROS ******************************/
#define SHR(a,b) ((a) >> (b))
#define ROTR(a,b) (((a) >> (b)) | ((a) << (32-(b))))
#define CH(x,y,z) (((x) & (y)) ^ (~(x) & (z)))
#define MAJ(x,y,z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))
#define EP0(x) (ROTR(x,2) ^ ROTR(x,13) ^ ROTR(x,22))
#define EP1(x) (ROTR(x,6) ^ ROTR(x,11) ^ ROTR(x,25))
#define SIG0(x) (ROTR(x,7) ^ ROTR(x,18) ^ SHR(x, 3))
#define SIG1(x) (ROTR(x,17) ^ ROTR(x,19) ^ SHR(x, 10))

/****************************** VARIABLES ******************************/
static const WORD k[64] = {
  0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
  0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
  0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
  0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
  0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
  0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
  0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
  0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
};

/*********************** FUNCTION DEFINITIONS ***********************/
void sha256_compress(SHA256_CTX *ctx)
{
        WORD a, b, c, d, e, f, g, h, i, j, t1, t2, m[64];

        for (i = 0, j = 0; i < 16; ++i, j += 4)
                m[i] = ((WORD)ctx->data[j] << 24)     | ((WORD)ctx->data[j + 1] << 16) |
                        ((WORD)ctx->data[j + 2] << 8) | ((WORD)ctx->data[j + 3]);
```

```
	for ( ; i < 64; ++i)
		m[i] = SIG1(m[i - 2]) + m[i - 7] + SIG0(m[i - 15]) + m[i - 16];

	a = ctx->state[0];
	b = ctx->state[1];
	c = ctx->state[2];
	d = ctx->state[3];
	e = ctx->state[4];
	f = ctx->state[5];
	g = ctx->state[6];
	h = ctx->state[7];

	for (i = 0; i < 64; ++i) {
		t1 = h + EP1(e) + CH(e,f,g) + k[i] + m[i];
		t2 = EP0(a) + MAJ(a,b,c);
		h = g;
		g = f;
		f = e;
		e = d + t1;
		d = c;
		c = b;
		b = a;
		a = t1 + t2;
	}

	ctx->state[0] += a;
	ctx->state[1] += b;
	ctx->state[2] += c;
	ctx->state[3] += d;
	ctx->state[4] += e;
	ctx->state[5] += f;
	ctx->state[6] += g;
	ctx->state[7] += h;
}

void sha256_init(SHA256_CTX *ctx)
{
	ctx->datalen = 0;
	ctx->bitlen = 0;
	ctx->state[0] = 0x6a09e667;
	ctx->state[1] = 0xbb67ae85;
	ctx->state[2] = 0x3c6ef372;
	ctx->state[3] = 0xa54ff53a;
	ctx->state[4] = 0x510e527f;
	ctx->state[5] = 0x9b05688c;
	ctx->state[6] = 0x1f83d9ab;
	ctx->state[7] = 0x5be0cd19;
}

void sha256_compute(SHA256_CTX *ctx, const BYTE data[], size_t len)
{
	WORD i;

	for (i = 0; i < len; ++i) {
		ctx->data[ctx->datalen] = data[i];
		ctx->datalen++;
		if (ctx->datalen == 64) {
			sha256_compress(ctx);
			ctx->bitlen += 512;
			ctx->datalen = 0;
		}
	}

	i = ctx->datalen;
```

```
	// Pad whatever data is left in the buffer.
	if (ctx->datalen < 56) {
		ctx->data[i++] = 0x80;
		while (i < 56)
			ctx->data[i++] = 0x00;
	}
	else {
		ctx->data[i++] = 0x80;
		while (i < 64)
			ctx->data[i++] = 0x00;
		sha256_compress(ctx);
		memset(ctx->data, 0, 56);
	}

	// Append to the padding the total message's length in bits.
	ctx->bitlen += ctx->datalen * 8;
	ctx->data[63] = ctx->bitlen;
	ctx->data[62] = ctx->bitlen >> 8;
	ctx->data[61] = ctx->bitlen >> 16;
	ctx->data[60] = ctx->bitlen >> 24;
	ctx->data[59] = ctx->bitlen >> 32;
	ctx->data[58] = ctx->bitlen >> 40;
	ctx->data[57] = ctx->bitlen >> 48;
	ctx->data[56] = ctx->bitlen >> 56;
	sha256_compress(ctx);
}

void sha256_convert(SHA256_CTX *ctx, BYTE hash[])
{
	int i;

	// Since this implementation uses little endian byte ordering and SHA uses big endian,
	// reverse all the bytes when copying the final state to the output hash.
	for (i = 0; i < 4; ++i) {
		hash[i]      = (ctx->state[0] >> (24 - i * 8));
		hash[i + 4]  = (ctx->state[1] >> (24 - i * 8));
		hash[i + 8]  = (ctx->state[2] >> (24 - i * 8));
		hash[i + 12] = (ctx->state[3] >> (24 - i * 8));
		hash[i + 16] = (ctx->state[4] >> (24 - i * 8));
		hash[i + 20] = (ctx->state[5] >> (24 - i * 8));
		hash[i + 24] = (ctx->state[6] >> (24 - i * 8));
		hash[i + 28] = (ctx->state[7] >> (24 - i * 8));
	}
}

void sha256(const BYTE data[], size_t len, BYTE hash[]) {
	SHA256_CTX ctx;
	sha256_init(&ctx);
	sha256_compute(&ctx,data,len);
	sha256_convert(&ctx,hash);
}
```

```
#include <stdio.h>
#include <memory.h>
#include <string.h>
#include "sha256.h"

void print_hash(unsigned char hash[])
{
   int idx;
   for (idx=0; idx < 32; idx++)
      printf("%02x",hash[idx]);
   printf("\n");
}
```

```
int main()
{
   unsigned char text1[]={"01234567879"},
                 text2[]={"0123456787901234567879012345678790123456787901234567879"},
                 hash[32];

   // Hash one
   sha256(text1, strlen(text1), hash);
   print_hash(hash);

   // Hash two
   sha256(text2, strlen(text2), hash);
   print_hash(hash);

   return 0;
}
```