galois

Lab: SHA512

File `SHA.cry` and `SHA512.cry` contain Cryptol functions that implement the `SHA512` hash plus create a digest for a given text string. Of specific interest is the function `sha M` in `SHA.cry` at the top which produces a digest from input text `M`. The function is defined like this:

```
sha :  {L} (2 * w >= width L) => [L] -> [digest_size]
sha M = take (join (SHA_2_Common' [ split x | x <- parse`{num_blocks L} (pad`{L} M) ]))
```

Run Cryptol and load `SHA512.cry` – `SHA.cry` is automatically loaded as well. An example run of this function is as follows:

```
SHA512> sha 0x00  // one byte
0xb8244d028981d693af7b456af8efa4cad63d282e19ff14942c246e50d9351d22704a802a71
c3580b6370de4ceb293c324a8423342557d4e5c38438f0e36910ee
```

Notice from the signature of `sha` that the output is `digest_size` bits wide and `digest_size` is defined as `512` in `SHA512.cry`. File `sha512.c` contains C code for producing a `SHA512` hash plus a digest. Adding a `main` like this:

```
int main(int args, char **argv) {
   int i=0;
   uint8_t *out = (uint8_t *)malloc(SHA512_DIGEST_LENGTH);
   uint8_t data[2];
   data[0] = 0x0;                // half byte
   data[1] = 0x0;                // half byte – two together are one byte
   out = SHA512(data, 1, out);   // One byte
   printf("0x");
   for (i=0 ; i < 64 ; i++) printf("%2x",out[i]);
   printf("\n");
}
```

and compiling allows one to display the digest given an input of one 0 byte like this:

```
[prompt]$ sha512
0xb8244d 28981d693af7b456af8efa4cad63d282e19ff14942c246e50d9351d22704a802a71
c358 b6370de4ceb293c324a8423342557d4e5c38438f0e36910ee
```

Observe that for the same input, the digest is the same. It is desired to show that the C function for computing the digest is functionally identical to the Cryptol “gold standard”. The Cryptol gold standard specification comes directly from the NIST/FIPS 180-4 publication which is presented as the background material for this lab. Several functions are defined in the documentation. These are defined in Cryptol and implemented in C. Start with the following functions on Page 11 of the publication.

$$\sum_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \quad (4.10)$$

$$\sum_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \quad (4.11)$$

$$\sigma_0^{\{512\}}(x) = ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x) \quad (4.12)$$

$$\sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x) \quad (4.13)$$

where $ROTR^n(x)$ is the rotate right (circular right shift) operation, where `x` is a `w`-bit word and `n` is an integer with $0 \le n < w$, and is defined by $ROTR^n(x) = (x >> n) \vee (x << w - n)$ and $SHR^n(x)$ is the right shift operation, where `x` is a `w`-bit word and `n` is an integer with $0 \le n < w$, and is defined by $SHR^n(x) = x >> n$. The Cryptol versions of these are in `SHA512.cry`:

```
SIGMA_0 x = (x >>> 28) ^ (x >>> 34) ^ (x >>> 39)
SIGMA_1 x = (x >>> 14) ^ (x >>> 18) ^ (x >>> 41)
sigma_0 x = (x >>> 1) ^ (x >>> 8) ^ (x >> 7)
sigma_1 x = (x >>> 19) ^ (x >>> 61) ^ (x >> 6)
```

The C implementation of these is the following in sha512.c:

```
static inline uint64_t Sigma0(uint64_t x) {
   return CRYPTO_rotr_u64((x), 28) ^ CRYPTO_rotr_u64((x), 34) ^
      CRYPTO_rotr_u64((x), 39);
}
static inline uint64_t Sigma1(uint64_t x) {
   return CRYPTO_rotr_u64((x), 14) ^ CRYPTO_rotr_u64((x), 18) ^
      CRYPTO_rotr_u64((x), 41);
}
static inline uint64_t sigma0(uint64_t x) {
   return CRYPTO_rotr_u64((x), 1) ^ CRYPTO_rotr_u64((x), 8) ^ ((x) >> 7);
}
static inline uint64_t sigma1(uint64_t x) {
   return CRYPTO_rotr_u64((x), 19) ^ CRYPTO_rotr_u64((x), 61) ^ ((x) >> 6);
}
```

where `CRYPTO_rotr_u64` is defined in `sha512.c`:

```
static inline uint64_t CRYPTO_rotr_u64(uint64_t value, int shift) {
   return (value >> shift) | (value << ((-shift) & 63));
}
```

Also, at this time consider the following function defined in the publication:

$$Ch(x, y, z) \quad = \quad (x \wedge y) \oplus (\neg x \wedge z)$$

In Cryptol this is:

```
Ch : [w] -> [w] -> [w] -> [w]
Ch x y z = (x && y) ^ (~x && z)
```

In C this is:

```
static inline uint64_t Ch(uint64_t x, uint64_t y, uint64_t z) {
   return ((x) & (y)) ^ ((~(x)) & (z));
}
```

Exercise 1:
Develop a SAW file for proving that the C functions above are equivalent to their corresponding Crypto specifications. Run `saw` on the file and report the result. ■

The next target is sha512_block_data_order because it depends on all of the functions above. The specification given in the publication is this:

Exercise 2:
Compile `sha-256.c` to `sha-256` from the command line. Run

```
sha-256 "Hello World Folks"
```

from the command line. Verify that the result is the same as above. ■

Exercise 3:
Create a function digest_in_bytes in `SHA256.cry` with signature

```
digest_in_bytes : {i} (fin i, 64 >= width (8*i)) => [i][8] -> [32][8]
```

That takes a message `msg` as input and outputs 32 bytes that is the digest of `msg`. Run

```
digest_in_bytes “Hello World Folks”
```

and verify the result matches the above except that the output is now a sequence of bytes. ■

Exercise 4:
The digest function of Cryptol does not require the length of the input message as input. The C function should not either. Write a C function named SHA256_Buf_Wrapper that takes, as input, the message, as a `char[]`, and the 32 byte digest array and inserts the digest of the message into the digest array. The prototype for the wrapper function is the following:

```
void SHA256_Buf_Wrapper(char *input, uint8_t digest[32]);
```

Then change `main` to look like this:

```
int main (int argc, char** argv) {
   // usage: sha256 <input>
   int i;
   uint8_t digest[32];
   SHA256_Buf_Wrapper(argv[1],digest);
   for (i=0 ; i < 32 ; i++)
      if (digest[i] < 16) printf("0%x",digest[i]);
         else printf("%x",digest[i]);
         printf("\n");
```

Run sha-256 “Hello World Folks” and verify the output is as above. ■

Exercise 5:
Following the pattern of the previous lab, construct a saw file that verifies that the C function `SHA256_Buf_Wrapper` on input `msg` produces the same output as the Cryptol function `digest_in_bytes` on `msg`. ■