Hermes: A Language for Lightweight Encryption

Torben Ægidius Mogensen

RC 2020

Background: Lightweight Encryption

- Lightweight: Meant to be fast.
- Symmetric key: Same key used for encryption and decryption.
- Used in embedded systems, browsers, etc.
- Must be resistant to side-channel attacks (memory, time, ...).
- Examples: AES, speck128, RC5, Blowfish, ...

Why Use a Reversible Language?

- Same code for encryption and decryption – you only need to write the encryption function, decryption comes for free.
- Leaves no garbage data (unless explicit) – partial protection from memory-based side-channel attacks.

Why Not Use Janus?

Janus is a reversible imperative language, and some experiments writing crypto functions have been done, but Janus has some limitations for this use:

- Not resistant to timing-based attacks: Janus-style conditionals and loops timing depend on data.
- No distinction between secret and public data, so no control over leaks.
- Only one integer type (of unspecified size). This can easily be fixed, though.

Introducing Hermes

- A reversible, imperative language borrowing elements from both Janus (reversible updates and procedures) and C (low-level bit manipulation, explicit integer sizes, syntax).
- Type system distinguishes secret and public data
- Operations on secret data use time that does not depend on the actual values.
- Rotates added as reversible updates.
- Formally specified type system and run-time semantics.
- Several implementations: reference interpreter (not secure, but follows semantics closely), compiler to C (assumes secure C compiler), compiler to WebAssembly (assumes secure compiler).

The Hermes Type System

Does information-flow analysis similar to binding-time analysis, but with additional restrictions to ensure reversibility and make execution time independent of the values of secret data. Details:

- Variables are by default secret, but can be declared public.
- Integers are 8-, 16-, 32-, or 64-bit unsigned.
- Arithmetic operations classified as constant or variable time. Operations on secret data must be constant time. Examples: +, <<, and $\wedge$ are constant time, / and % are variable time.
- Array sizes and indices must be public (since caching may affect lookup time).
- Loop counters and loop bounds must be public.
- Aliasing restrictions ensure reversibility of updates and parameter passing.

Formal type system specification in paper.

Control Structures

Limited to those commonly used in light-weight encryption:

- For loop (public bounds and counter). Arbitrary reversible updates of counter (as long as no secret data involved).
- Conditional updates and swaps. Uses bitmasks to ensure constant time, so they can be used on secret data.
- Reversible procedure calls (`call` and `uncall`) with call-by-reference parameters. Aliasing restrictions to ensure reversibility.

Example: RC5 Core in C (Encryption Only)

```
#define ROL(x,r) ((x<<r)|(x>>(32-r)))

void RC5_ENCRYPT(WORD *pt, WORD *ct)
{
   WORD i, A=pt[0]+S[0], B=pt[1]+S[1];

   for(i = 1; i <= 12; i++)
   {
      A = ROL(A ^ B, B) + S[2*i];
      B = ROL(B ^ A, A) + S[2*i + 1];
   }
   ct[0] = A; ct[1] = B;
}
```

- Rotate not built-in, so defined in macro.
- Not obviously reversible.

Example: RC5 Core (Hermes)

```
rc5(u32 ct[], u32 S[])
{
  u32 A, B;
  A <-> ct[0]; B <-> ct[1];
  A += S[0]; B += S[1];
  for(i=2; size S) {
    A ^= B; A <<= B; A += S[i];
    B <-> A;
    i++;
  }
ct[0] <-> A; ct[1] <-> B;
}
```

- Rotate (<<=) is built into Hermes.
- Obviously reversible (only reversible operations used, encrypted text replaces plaintext).
- Loop is "rolled" by using swap.

Example: Speck128 (Hermes)

```
speck128(u64 ct[], u64 K[])
{  /* with on-the-fly key expansion */
   u64 y, x, b, a;
   y <-> ct[0]; x <-> ct[1]; b += K[0]; a += K[1];

   call Rs(x, y, b);
   for (i=0; 32) {
     call Rp(a, b, i); i++; call Rs(x, y, b);
   }
   /* key un-expansion */
   for (i=32; 0) { i--; uncall Rp(a, b, i); }
   y <-> ct[0]; x <-> ct[1]; b -= K[0]; a -= K[1];
}

Rs(u64 x, u64 y, secret u64 k)
{ x >>= 8; x += y; x ^= k; y <<= 3; y ^= x; }

Rp(u64 x, u64 y, public u64 k)
{ x >>= 8; x += y; x ^= k; y <<= 3; y ^= x; }
```

Limitations and Proposed Extensions

Some limitations were encountered when using Hermes to implement some standard cyphers:

- Speck128 required two copies of a procedure (secret and public parameters). Proposed extension: Read-only parameters.
- AES uses secret values to index arrays. Proposed extension: Cached arrays.
- AES uses if-then. Emulated using loop: Counter starts at entry condition and ends at 0, subtract exit condition from counter after loop body.
- Key expansion (AES, Blowfish) is not always reversible. Solution: Use garbage array.

Other Proposed Extensions / Future Work

- Sized Boolean types (guaranteed to be all 0 bits or all 1 bits).
- More control structures such as if-then-else. Public control only.
- Call-by-value-result for scalar parameters. Better performance and indistinguishable from call-by-reference due to aliasing restrictions.
- Global variables. Need to extend aliasing restrictions.
- Partial evaluation to eliminate conditional jumps and enable compile-time index and zero checks.
- Larger integer sizes.