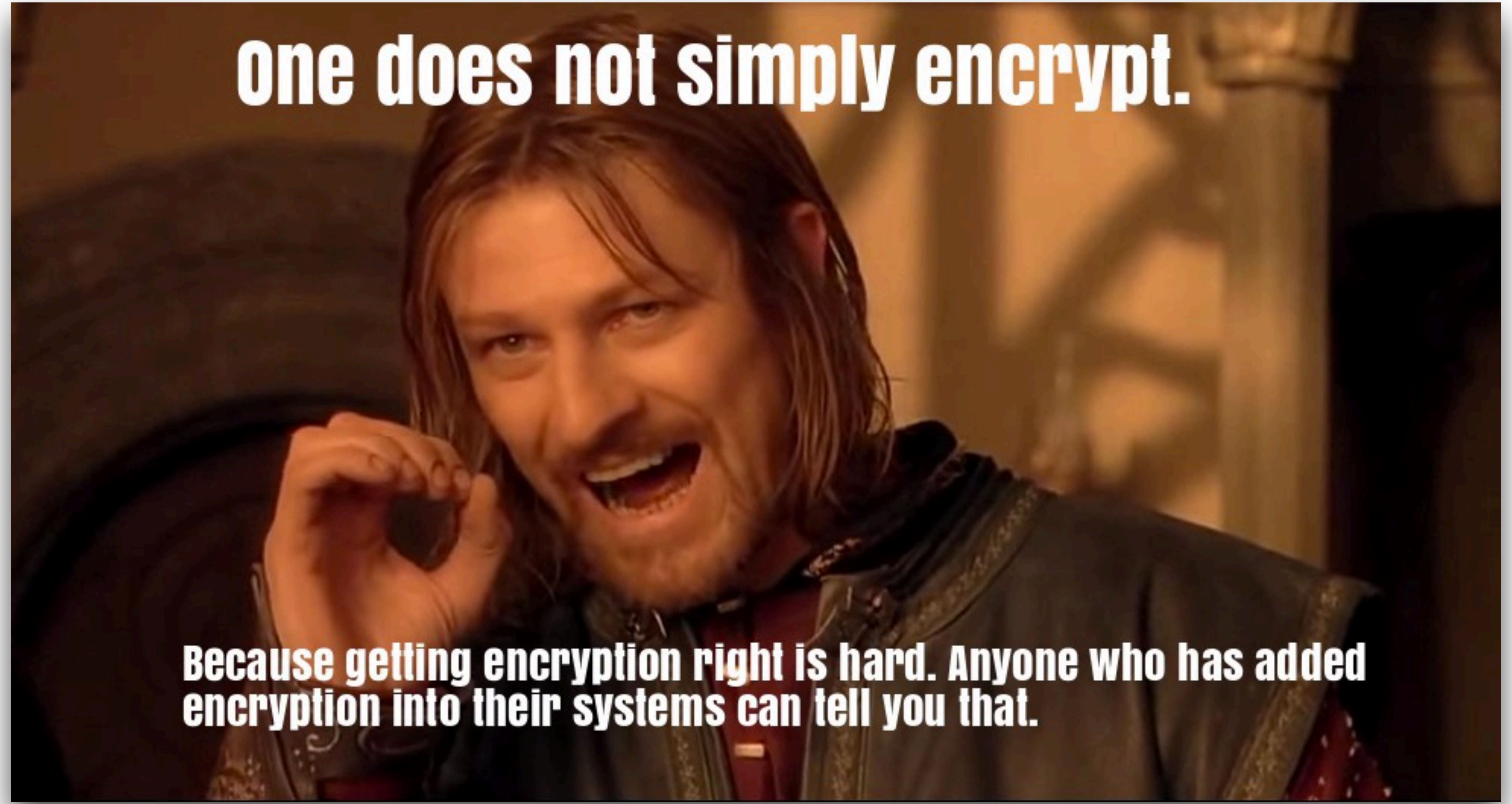
CRYPTAL

Sammy Tbeile | Jaewan Bahk | Michail Oikonomou
Carolina Almirola | Rahul Kapur

Overview


One does not simply encrypt.
Because getting encryption right is hard. Anyone who has added encryption into their systems can tell you that.

Motivation

- Combined interest in the fields of security and cryptography.
- No well-documented or straightforward languages/packages that help alleviate the pains of modular arithmetic and complicated encryption schemes for users.
- Given the growing demand for more secure systems, a language designed for ease of implementation of encryption schemes is a valuable addition to the field of computer science and security engineering.

About Our Language

- C-like syntax
- Compiles to LLVM
- Built-in types for modular integers and large numbers:
    - Gems: The gem type consists of a value and a modular value. All operations performed on a gem are done as modular arithmetic.
    - Lattices: Built-in representation for large numbers.
    - Integers: The same integers we know and love from C.
- Mixed operations between gem, int, and lattice make arithmetic straightforward and remove burden from users of keeping track of numerical limits.

Special Features

- Modular Arithmetic:
    - Arithmetic operations on gems maintain modular state
    - Addition, Subtraction, Power, Multiplication, Division
- Modular Inverse:
    - Intuitive syntax for obtaining the modular inverse of a number
        - example:

```
gem a = (3, 5)
gem b = !a
print_gem(b)

>> 2
```

- Built-in MD5 Hashing
- Print:
    - `print_gem` and `print_lat` allow for direct printing to stdout of gem and lattice values.

How a BN becomes a Gem

- We use openssl's BIGNUM library to implement arithmetic between gems and lattices.
- Modular arithmetic operations are defined in crypto_arith.c
- codegen.ml uses these functions

The Game Plan

Proposal
+
LRM

Testing

Hello World!

First drafts of parser, scanner, ast, semant, & codegen

Testing

Expressions
+
Built-in Types

Implementation of expressions and statements
+
Operations on built-in types

Testing

Encryption Schemes

Implement some well known schemes using our new language

Division of Labour

Roles/Responsibilities

- **Sammy (System Architect):**
  - Integration of openSSL and BN in Codegen.
  - Implementation of expressions and built-in functions.
- **Jaewan (Language Guru/Tester):**
  - Semantic checking and language documentation and specification.
  - Testing
  - Made the logo!
- **Michail (System Architect/Tester):**
  - Implementation of expressions and statements and built-in functions
  - Testing for continuous integration.
- **Carolina (Manager):**
  - Semantic checking for mathematical expressions and statements.
  - Language documentation and Final Report.
- **Rahul Kapur (Tester):**
  - Test suite and continuous integration.

And now for some demos…

chinese_remainder_thm.crp

```
int main(){
  int a;
  int mod_a;
  int b;
  int mod_b;

  gem x;
  lat x_scratch;

  gem y;
  lat y_scratch;

  gem z;

  a = 5;
  mod_a = 7;

  b = 4;
  mod_b = 8;

  x = (2, mod_a);
  y = (3, mod_b);

  x_scratch = !x;
  y_scratch = !y;

  z = (x_scratch * a * mod_b + y_scratch * b * mod_a, (mod_a * mod_b));
  print_gem(z);
}
```

diffie-hellman.crp

```
gem sign_alice_exponent(gem a) {
  lat alice_secret_exponent;
  gem alice_message_signed;

  alice_secret_exponent = 3;

  alice_message_signed = a ** alice_secret_exponent;

  return alice_message_signed;
}
```

```
gem sign_bob_exponent(gem b) {
  lat bob_secret_exponent;
  gem bob_message_signed;

  bob_secret_exponent = 4;

  bob_message_signed = b ** bob_secret_exponent;

  return bob_message_signed;
}
```

```
int main() {
  lat PRIME;
  lat NUM;

  gem alice_message;
  gem bob_message;

  PRIME = 15485863;
  NUM = 32452843;

  alice_message = (NUM, PRIME);
  alice_message = sign_alice_exponent(alice_message);

  bob_message = (NUM, PRIME);
  bob_message = sign_bob_exponent(bob_message);


  if (sign_alice_exponent(bob_message) == sign_bob_exponent(alice_message)) {
    print("Diffie-Hellman Key Exchange Successful");
  } else {
    print("Diffie-Hellman Key Exchange Failed");
  }

  return 0;
}
```

euclidean_algorithm.crp

```
lat gcd(lat a, lat b) {
  gem rem;
  rem = (a, b);
  while (rem != 0) {
    a = b;
    b = rem;
    rem = (a, b);
  }
  return b;
}

int main() {
    lat a;
    lat b;
    lat g;
    a = 10;
    b = 50;
    g = gcd(a, b);
    print_lat(g);
}
```

hash-md5.crp

```
int main() {
    lat a;

    a = hash_md5("hello_world");
    print_lat(a);
}
```