Verified Post-Quantum Cryptography

•••

Milestone 1

Tom Arnold <tca4384@rit.edu>

Switch from FStar -> LiquidHaskell

- Abandoning FStar.
  - No documentation for standard library or comments in code to explain how things work.
  - HACL less useful than expected (no general purpose matrices).
  - General unfamiliarity with OCaml.

- LiquidHaskell
  - Extension to Haskell that adds refinement types verified by Z3 SMT solver (like FStar).
  - Comments in code to specify refinements.
  - Plugin to Haskell compiler integrates seamlessly with normal Haskell projects.
  - Better documentation than FStar / familiarity with Haskell.
  - https://ucsd-progsys.github.io/liquidhaskell/

Results

- Implemented McEliece w/ Hamming (7,4) code in LiquidHaskell.
    - Key generation is manual, still need to write code to generate random matrices and find their inverses.
    - ~140 lines of code + ~70 lines of refinements
    - Syndrome decoding took at least an hour to create a working refinement.
    - Errors caught:
        - Numerous off-by-one errors (>0 vs >=0).
        - Incompatible matrix dimensions. MxK · KxN
    - https://bitbucket.org/Tom9729/csci788/src/2c13bb942e006bf077fba4e6a9106b1023a611bf/lib/

```
[ 9 of 12] Compiling McElieceHamming  ( /home/tom/wkspace/csci788/lib/McElieceHamming.hs

**** LIQUID: SAFE (280 constraints checked) ****************************************
```

Roadmap For Milestone 2

- Review and implement Goppa code version of McEliece.
    - Galois fields and Patterson's decoding algorithm.
- Finish implementing key generation.
    - Random scramble/permutation matrices.
    - Calculating matrix inverses.

McEliece Public Key Encryption (PKE) Algorithm

- Linear algebra and coding theory.
- Based on a linear code (typically Goppa codes).
- Keys are matrices, messages are vectors, main operation is multiplication.

Bob: Shares public key SGP -> G'

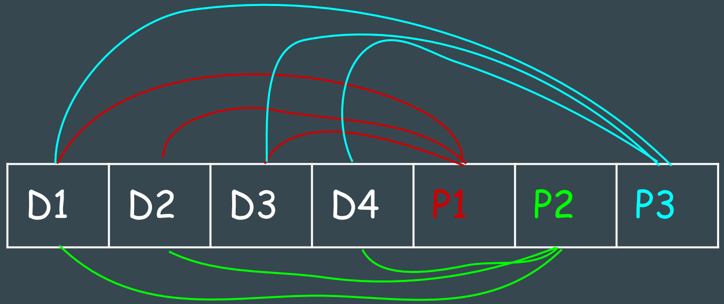
Alice: Encrypt message xG' + e -> y

Bob: Decrypts message yP^{-1} -> xSG + e' -> xSG (via syndrome decoding) -> xS (via clever choice of G) -> x**S**S^{-1} **-> x**

[1] McEliece Cryptosystem. <http://www-math.ucdenver.edu/~wcherowi/courses/m5410/ctcmcel.html>

Linear Codes

- Encoding data to detect and correct transmission errors (bit flips).
- Linear code in McEliece is pluggable, for M1 we used a simple Hamming code.
- Hamming (7,4) code:
  - Encodes 4 bits of data with 3 even parity bits, can correct up to 1 error in the data.



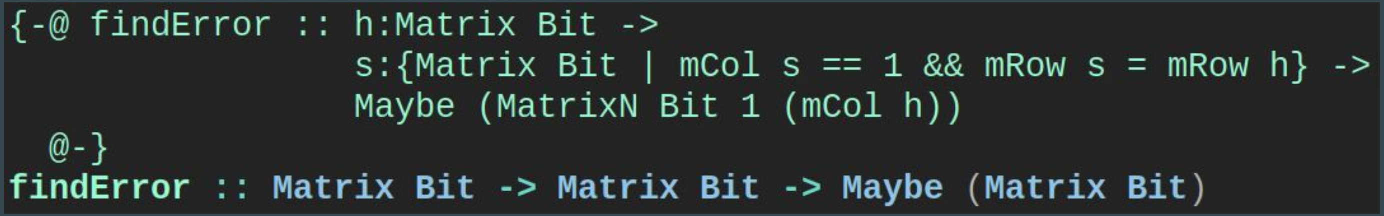
D1 + D2 + D3 + P1 = 0   e.g. 1 + 0 + 0 + 1 = 0

...

[2] Hamming Code: A Matrix Approach <https://www.ece.unb.ca/tervo/ee4253/hamming2.shtml>

Refinement Types

- Combining types (Int, String, Bool) with predicates (>, =, AND, OR) to check inputs/outputs to functions.
- Use cases:
  - Total functions (function terminates)
  - Constraining values
  - Size-aware API



Refinement type ->

```
{-@ findError :: h:Matrix Bit ->
                 s:{Matrix Bit | mCol s == 1 && mRow s = mRow h} ->
                 Maybe (MatrixN Bit 1 (mCol h))
  @-}
```

Haskell signature ->

```
findError :: Matrix Bit -> Matrix Bit -> Maybe (Matrix Bit)
```

[3] Programming with Refinement Types. <https://ucsd-progsys.github.io/liquidhaskell-tutorial/Tutorial_01_Introduction.html>

References

1. McEliece Cryptosystem. <http://www-math.ucdenver.edu/~wcherowi/courses/m5410/ctcmcel.html>
2. Hamming Code: A Matrix Approach. <https://www.ece.unb.ca/tervo/ee4253/hamming2.shtml>
3. Programming with Refinement Types. <https://ucsd-progsys.github.io/liquidhaskell-tutorial/Tutorial_01_Introduction.html>