CRYLOCK:
MINILOCK IN CRYPTOL

DAVE ARCHER, THOMAS DUBUISSON

MAY, 2015

WHAT IS CRYPTOL?

CRYPTOL

- A DSL for algorithms over generic bitstreams
- A tool for
  - Executing the specified algorithms
  - Stating, verifying, and proving algorithm properties
  - Verifying equivalence of implementations (C, Java, MATLAB...)
  - Generating implementations (LLVM bytecode, VHDL)
- Goal: Cryptol specification is verifiable by inspection against a mathematical definition

IN THE WILD

- Intelligence communities
- Corporate crypto experts
- Crypto research and education community
- Over 10 years of active use

EVERYTHING IS BITS

```
Cryptol> :type 0x1234
0x1234 : [16]
Cryptol> :type [1,2,3,4,5]
[1, 2, 3, 4, 5] : {a} (fin a, a >= 3) => [5][a]
```

FUNCTIONAL, STRONGLY TYPED

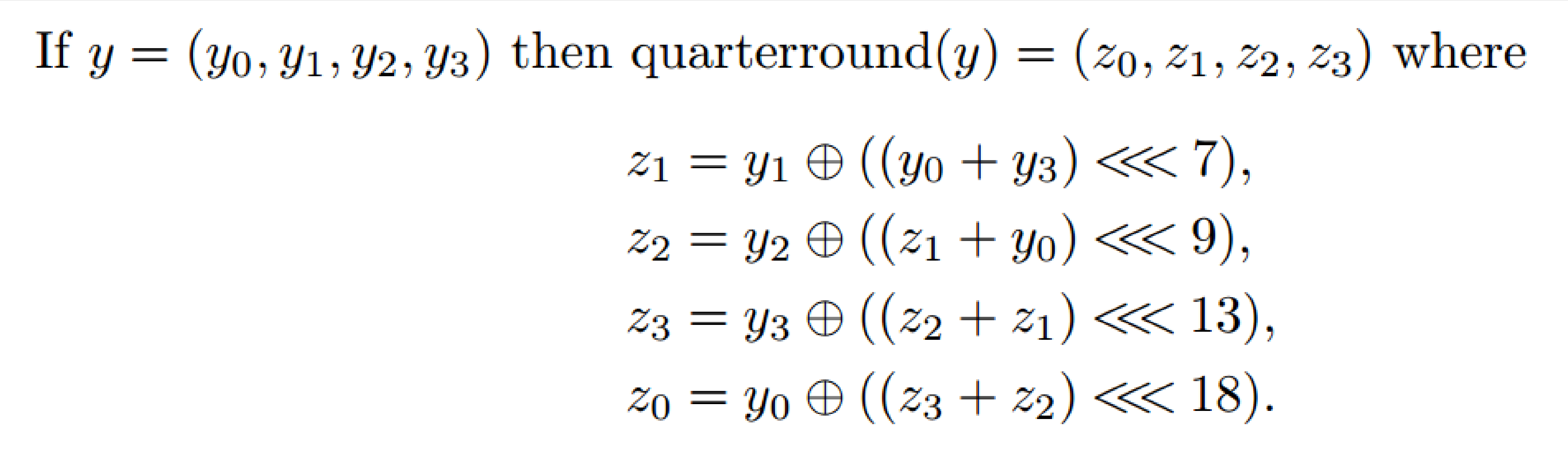
```
Cryptol> :t split
split : {parts, each, a} (fin each)
     => [parts * each]a -> [parts][each]a
```

```
littleEndian word = reverse (split `{each=8} word)
```

```
littleEndian word = reverse (split `{parts=4} word)
```

```
littleEndian : [32] -> [4][8]
littleEndian word = reverse (split word)
```

CRYPTOL <> SPEC



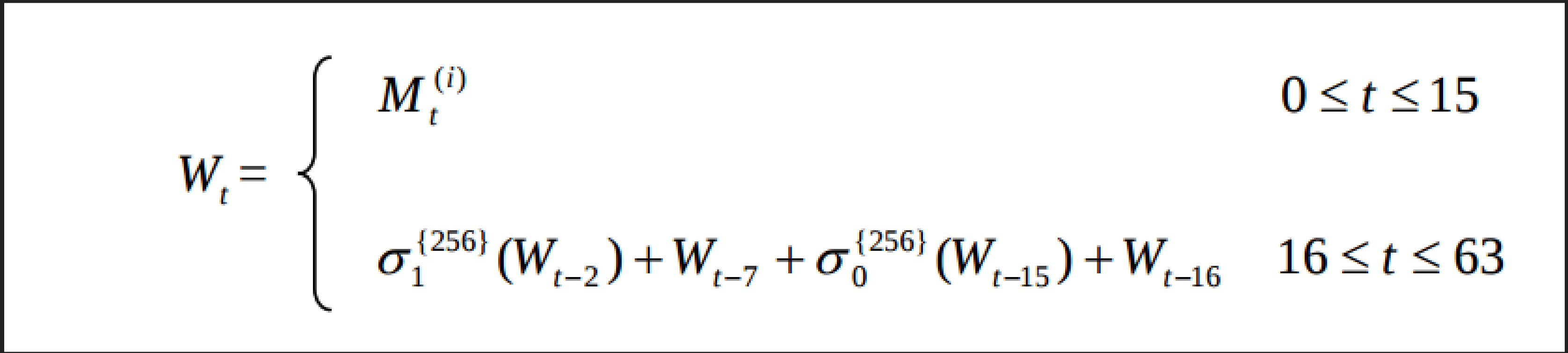
If $y = (y_0, y_1, y_2, y_3)$ then quarterround$(y) = (z_0, z_1, z_2, z_3)$ where

$$\begin{aligned} z_1 &= y_1 \oplus ((y_0 + y_3) \lll 7), \\ z_2 &= y_2 \oplus ((z_1 + y_0) \lll 9), \\ z_3 &= y_3 \oplus ((z_2 + z_1) \lll 13), \\ z_0 &= y_0 \oplus ((z_3 + z_2) \lll 18). \end{aligned}$$

```
quarterround : [4][32] -> [4][32]
quarterround [y0, y1, y2, y3] = [z0, z1, z2, z3]
  where
    z1 = y1 ^ ((y0 + y3) <<< 0x7)
    z2 = y2 ^ ((z1 + y0) <<< 0x9)
    z3 = y3 ^ ((z2 + z1) <<< 0xd)
    z0 = y0 ^ ((z3 + z2) <<< 0x12)
```

CRYPTOL <> SPEC

```
SHA256MessageSchedule : [16][32] -> [64][32]
SHA256MessageSchedule M = W where
    W = M #
        [ s1 (W@(t-2)) + W@(t-7) + s0 (W@(t-15)) + W@(t-16)
              | t <- [16 .. 63]  ]
```



$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

SHA256

EXPRESSIVE, SPECIALIZED

```
while (b >= 64) {
  crypto_core_salsa20(x,z,k,sigma);
  for (i = 0; i < 64; i++) c[cpos+i] = m[mpos+i] ^ x[i];
  u = 1;
  for (i = 8; i < 16; i++) {
    u = u + (z[i] & 0xff) | 0;
    z[i] = u & 0xff;
    u >>>= 8;
  }
  b -= 64;
  cpos += 64;
  mpos += 64;
}
```

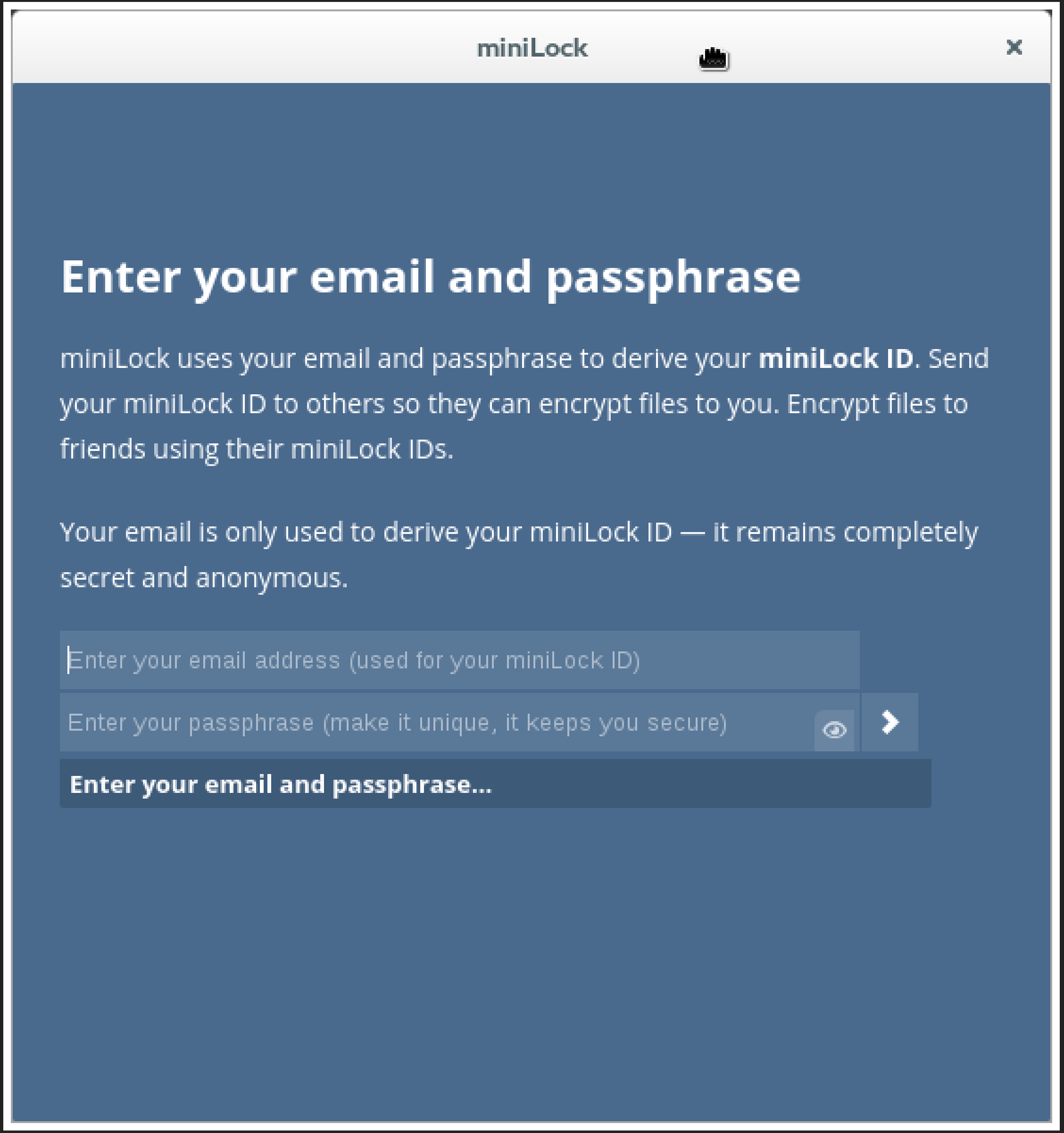
```
salsa = join [ Salsa20(k, v#(split i)) | i <- [0, 1 ...]]
c     = m ^ take salsa
```

CRYPTOL IS NOT...

- A protocol description language
  - No means of expressing communication between parties
- A protocol verification suite
  - No reasoning about concurrent actions by multiple parties
- A general-purpose programming language
  - No data management appropriate for general data processing

MINILOCK PROJECT

LOG IN



Enter your email and passphrase

miniLock uses your email and passphrase to derive your **miniLock ID**. Send your miniLock ID to others so they can encrypt files to you. Encrypt files to friends using their miniLock IDs.

Your email is only used to derive your miniLock ID — it remains completely secret and anonymous.

Enter your email address (used for your miniLock ID)

Enter your passphrase (make it unique, it keeps you secure)

Enter your email and passphrase...

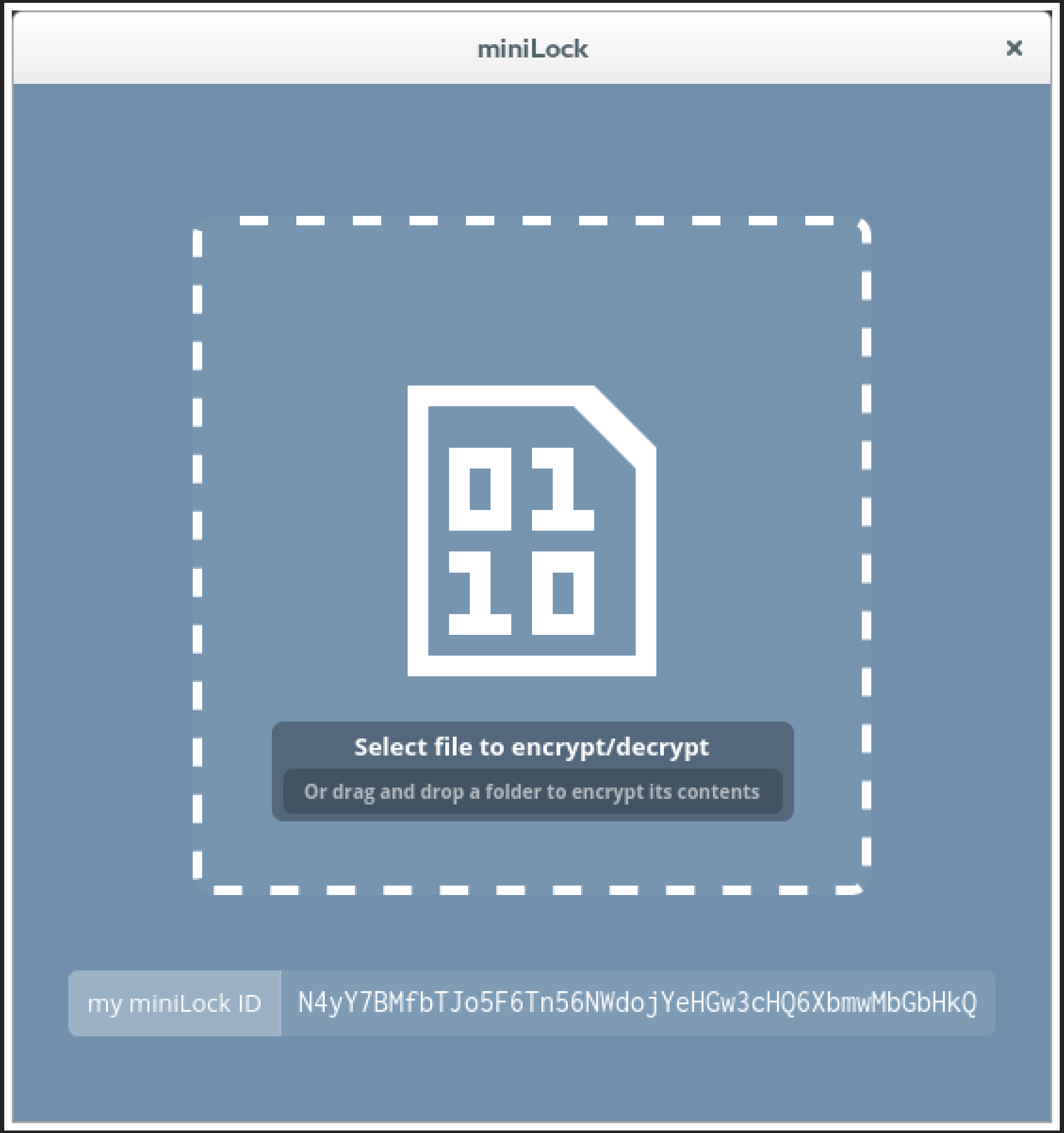
KEYS

private key = SCrypt (Blake2s(password))

public key = Curve25519(private, 9)

minilock ID = base58(public || blake2s_1(public))

UI

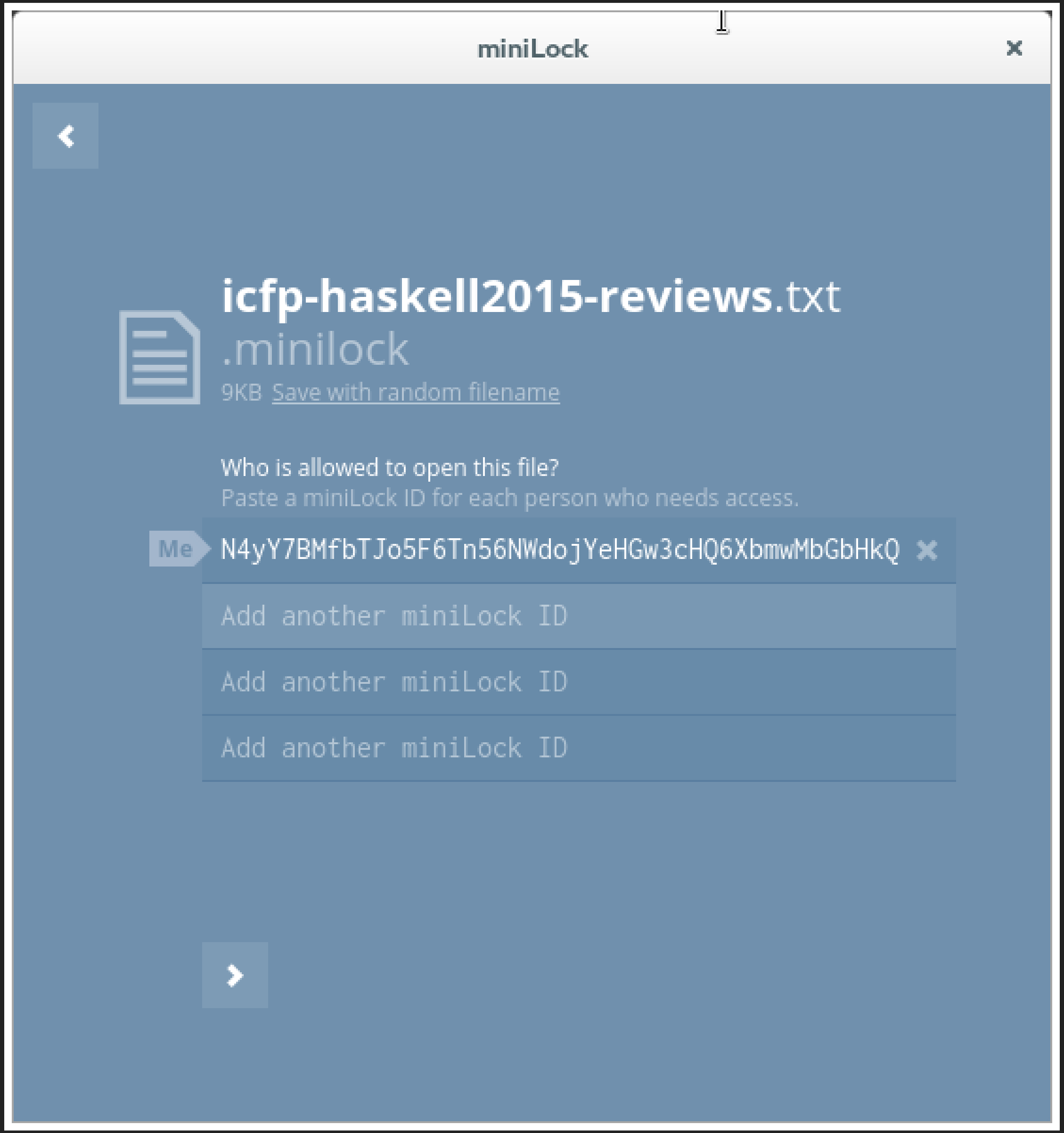


Select file to encrypt/decrypt

Or drag and drop a folder to encrypt its contents

my miniLock ID: N4yY7BMfbTJo5F6Tn56NWdojYeHGw3cHQ6XbmwMbGbHkQ

ENCRYPTION



icfp-haskell2015-reviews.txt
.minilock

9KB Save with random filename

Who is allowed to open this file?
Paste a miniLock ID for each person who needs access.

Me N4yY7BMfbTJo5F6Tn56NWdojYeHGw3cHQ6XbmwMbGbHkQ

Add another miniLock ID

Add another miniLock ID

Add another miniLock ID

MINILOCK OPERATIONS

$k \leftarrow \{0, 1\}^{256}$; $n \leftarrow \{0, 1\}^{192}$

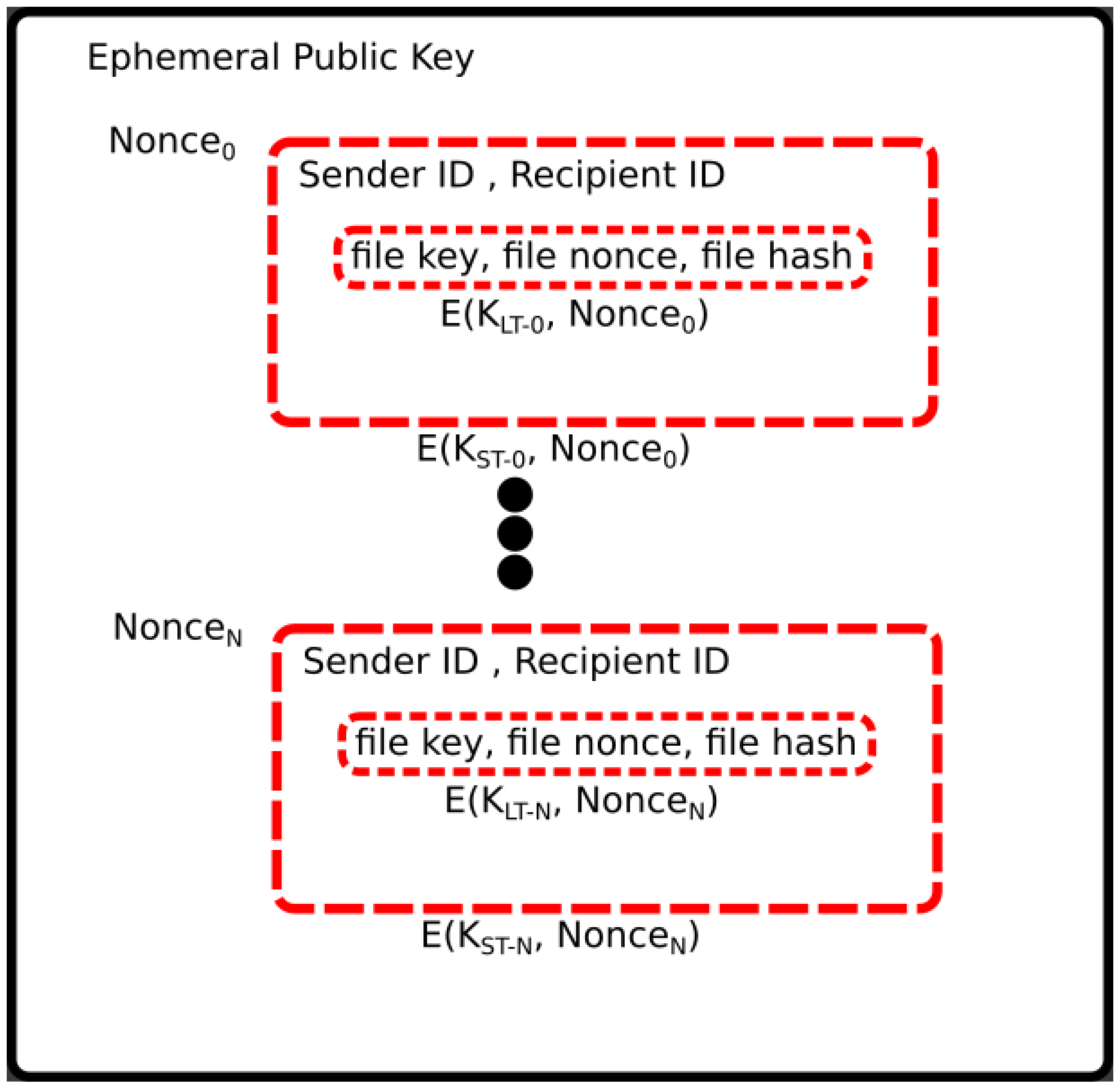
ciphertext = XSalsa20+Poly1305(k, n, file)

For each recipient:

- K_{LT} = Curve25519(Sender, Receiver)
- RecipInfo = IDs || $E_{K_{\mathrm{LT}}}$(k || n || blake2s(file))
- K_{ST} = Curve25519(Ephemeral, Receiver)
- RI = $E_{K_{\mathrm{st}}, \mathrm{nonce}}$(RecipInfo)

Header = Ephemeral Public Key || Nonce$_{1 \ldots N}$ || $RI_{1 \ldots N}$

MINILOCK FORMAT



Ephemeral Public Key

Nonce_0

Sender ID , Recipient ID

file key, file nonce, file hash

$E(K_{LT\text{-}0}, \text{Nonce}_0)$

$E(K_{ST\text{-}0}, \text{Nonce}_0)$

Nonce_N

Sender ID , Recipient ID

file key, file nonce, file hash

$E(K_{LT\text{-}N}, \text{Nonce}_N)$

$E(K_{ST\text{-}N}, \text{Nonce}_N)$

CRYPTOL SPEC

- About two weeks, including time spent with tooling
- ~1500 lines of primitives
- ~200 lines of minilock-specific glue code
- Interoperable
  - File constructed with crylock can be opened by Minilock

grain of salt

Original (javascript) ~4000 lines and 700 lines respectively.

PRIMITIVES USED

- Key derivation
  - SHA256
  - HMAC
  - PBKDF2
  - SCrypt
- Encapsulation
  - NaCl's crypto_box
    - XSalsa
    - Poly1305
    - Curve25519
  - Blake2s
- Formatting
  - Base64, Base58

DEMONSTRATION

BRASS TACKS

PROPERTIES

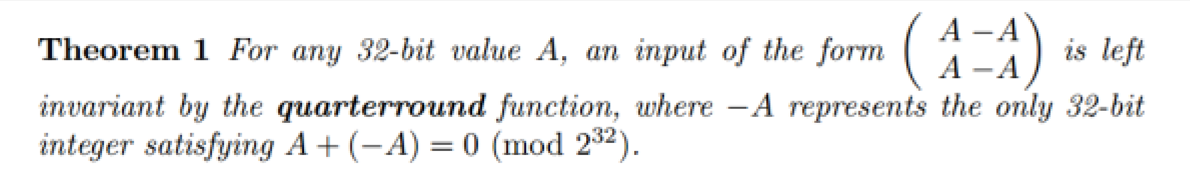
```
property Salsa20_doubleround_has_no_collisions x1 x2 =
  x1 == x2 || doubleround x1 != doubleround x2
```

```
Salsa20> :check Salsa20_doubleround_has_no_collisions
Using random testing.
passed 100 tests.
Coverage: 0.00% (100 of ...)
```

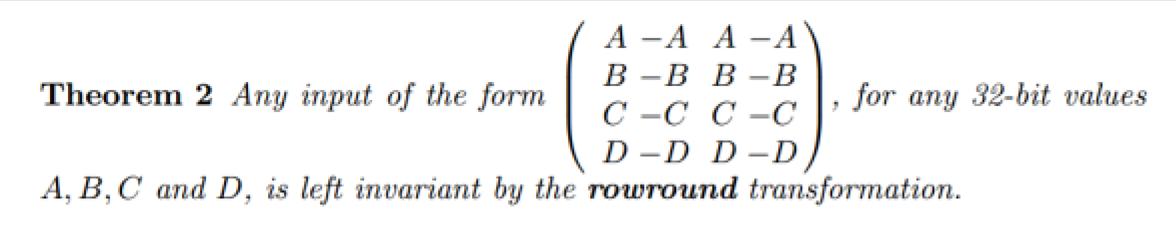
```
Salsa20> :exhaust Salsa20_doubleround_has_no_collisions
Using exhaustive testing.
  0%Ctrl-C
```

```
Salsa20> :set prover=z3
Salsa20> :prove Salsa20_doubleround_has_no_collisions
Q.E.D.
```

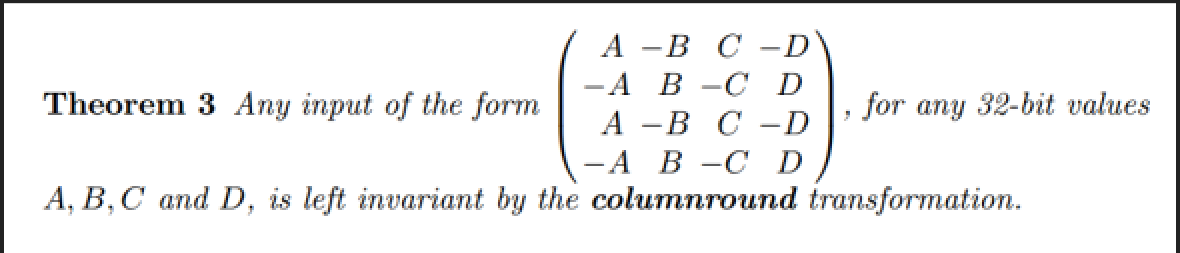
BRASS TACKS: PROPERTIES



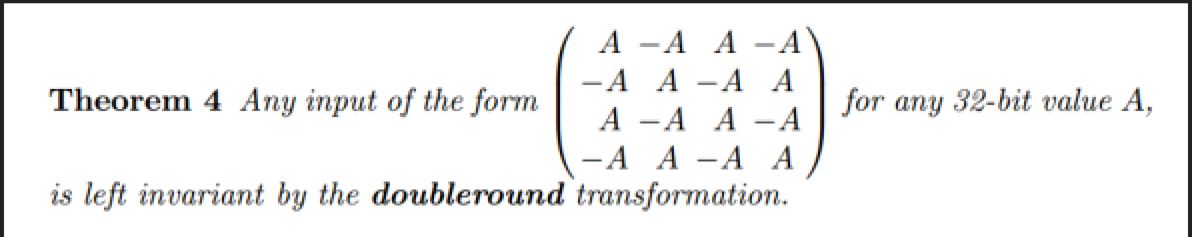
Theorem 1 *For any 32-bit value A, an input of the form* $\begin{pmatrix} A & -A \\ A & -A \end{pmatrix}$ *is left invariant by the* **quarterround** *function, where $-A$ represents the only 32-bit integer satisfying* $A + (-A) = 0 \pmod{2^{32}}$.



Theorem 2 *Any input of the form* $\begin{pmatrix} A & -A & A & -A \\ B & -B & B & -B \\ C & -C & C & -C \\ D & -D & D & -D \end{pmatrix}$*, for any 32-bit values A, B, C and D, is left invariant by the* **rowround** *transformation.*



Theorem 3 *Any input of the form* $\begin{pmatrix} A & -B & C & -D \\ -A & B & -C & D \\ A & -B & C & -D \\ -A & B & -C & D \end{pmatrix}$*, for any 32-bit values A, B, C and D, is left invariant by the* **columnround** *transformation.*



Theorem 4 *Any input of the form* $\begin{pmatrix} A & -A & A & -A \\ -A & A & -A & A \\ A & -A & A & -A \\ -A & A & -A & A \end{pmatrix}$ *for any 32-bit value A, is left invariant by the* **doubleround** *transformation.*

BRASS TACKS: PROPERTIES

```
property theorem1 a = quarterround [a, -a, a, -a] == [a,-a,a,-a]
```

```
property theorem2 a b c d = rowround val == val
    where val    = [a,-a,a,-a
                   ,b,-b,b,-b
                   ,c,-c,c,-c
                   ,d,-d,d,-d]
```

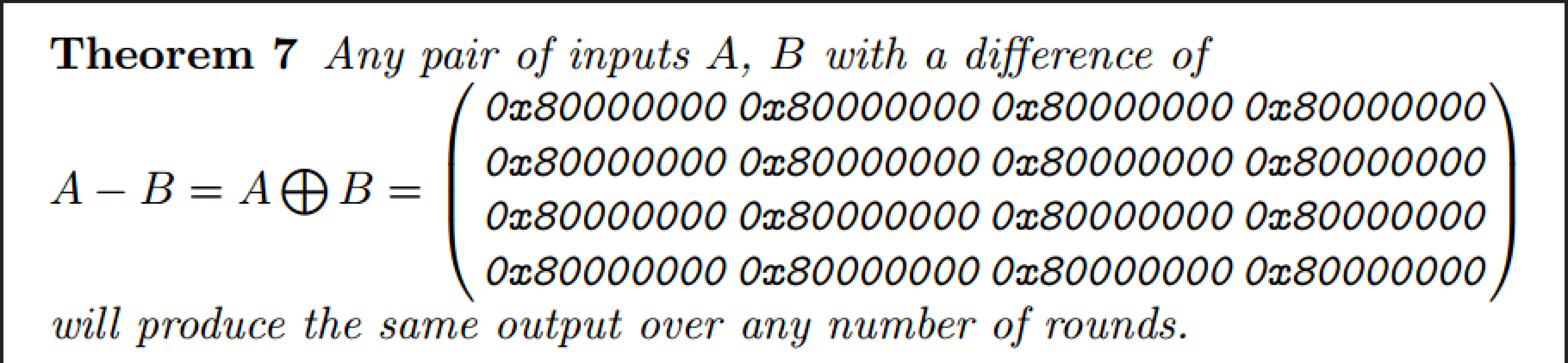
```
property theorem3 a b c d = columnround val == val
    where val    = [a,-b,c,-d
                   ,-a,b,-c,d
                   ,a,-b,c,-d
                   ,-a,b,-c,d]
```

```
property theorem4 a = doubleround val == val
    where val    = [a,-a,a,-a
                   ,-a,a,-a,a
                   ,a,-a,a,-a
                   ,-a,a,-a,a]
```

BRASS TACKS: PROPERTIES

```
Salsa20> :set prover=any
Salsa20> :prove theorem1
Q.E.D.
Salsa20> :prove theorem2
Q.E.D.
Salsa20> :prove theorem3
Q.E.D.
Salsa20> :prove theorem4
Q.E.D.
```

BRASS TACKS: NO RESULTS



Theorem 7 *Any pair of inputs A, B with a difference of*

$$A - B = A \bigoplus B = \begin{pmatrix} 0x80000000 & 0x80000000 & 0x80000000 & 0x80000000 \\ 0x80000000 & 0x80000000 & 0x80000000 & 0x80000000 \\ 0x80000000 & 0x80000000 & 0x80000000 & 0x80000000 \\ 0x80000000 & 0x80000000 & 0x80000000 & 0x80000000 \end{pmatrix}$$

will produce the same output over any number of rounds.

```
property theorem7 a b =
    a ^ b != diff || Salsa20Words a == Salsa20Words b
 where
  diff = [ 0x80000000 | _ <- [0..15]]
```

Last minute result: Boolector did terminate after an unimpressive week+ of computation.

BRASS TACKS: POSITIVE RESULTS

```
property speckKeyExpansionInjective x y =
   x == y || speckKeyExpansion x != speckKeyExpansion y
```

```
SimonAndSpeck> :set prover=boolector
SimonAndSpeck> :prove speckKeyExpansionInjective
Q.E.D.
```

```
property simon_ident_128_128 k p =
    simonD_128_128 k (simonE_128_128 k p) == p
```

```
SimonAndSpeck> :set prover=any
SimonAndSpeck> :prove simon_ident_128_128
Q.E.D.
```

BRASS TACKS: NEGITIVE RESULTS

```
property ZUC_isResistantToCollisionAttack k iv1 iv2 =
    iv1 == iv2 ||
    InitializeZUC (k, iv1) @ 1 != InitializeZUC (k, iv2) @ 1
```

```
Main> :prove ZUC_isResistantToCollisionAttack
ZUC_isResistantToCollisionAttack 0x1200000000700000000c0000085006
                                 0x0e008f00ff0000000000ff0000ed004
                                 0x96008f00ff0000000000ff0000ed004
    = False
```

BEYOND CRYPTOL

SOFTWARE ANALYSIS WORKBENCH

- SAWScript: a special-purpose scripting language
  - Construct, manipulate, and query mathematical models of software semantics
  - Supports LLVM, JVM, Cryptol
  - Proofs of properties using automated provers
  - Compositional proof techniques
- Represents 'Term's in a dependently typed lambda calculus
- Interface via an interactive REPL or batch scripts

SAW: TRIPLE DES EXAMPLE

```
m       <- cryptol_load "DES.cry";
enc     <- define "enc" {{ m::DES.encrypt }};
dec     <- define "dec" {{ m::DES.decrypt }};
dec_enc <- time (prove abc {{ \k m -> dec k (enc k m) == m }});
enc_dec <- time (prove abc {{ \k m -> enc k (dec k m) == m }});
letss = simpset [dec_enc, enc_dec];
let{{
  enc3 k1 k2 k3 msg = enc k3 (dec k2 (enc k1 msg))
  dec3 k1 k2 k3 msg = dec k1 (enc k2 (dec k3 msg))
  dec3_enc3 k1 k2 k3 msg = dec3 k1 k2 k3 (enc3 k1 k2 k3 msg) == ms
}};
time (prove do {simplify ss; abc; } {{ dec3_enc3 }});
```

```
Valid
Time: 4.694s
Valid
Time: 4.718s
Valid
Time: 0.003s
```

SOFTWARE ANALYSIS WORKBENCH: COMPOSITIONAL PROOFS

```
property theorem7 a b =
    a ^ b != diff || Salsa20Words a == Salsa20Words b
 where
  diff = [ 0x80000000 | _ <- [0..15]]
```

```
t1_po <- prove_print abc {{ theorem1 }};
...
t5_po <- prove_print abc {{ theorem5 }};

let ss = simpset [ t1_po, t2_po, t3_po, t4_po, t5_po ];
prove_print do { simplify ss; abc ; }
  {{ \a -> ( (Salsa20Words a == Salsa20Words (a ^ diff))
                    where diff = [ 0x80000000 | _ <- [0..15] )
  }};
```

```
Valid
Valid
Valid
Valid
Valid
Proving Theorem 7
Valid
```

EQUIVALENCE CHECKING

```
void core(u8 *out,const u8 *in,const u8 *k,const u8 *c) {
    u32 w[16],x[16],y[16],t[4];
    int i,j,m;

    FOR(i,4) {
        x[5*i] = ld32(c+4*i);
        x[1+i] = ld32(k+4*i);
        x[6+i] = ld32(in+4*i);
        x[11+i] = ld32(k+16+4*i);
    }

    FOR(i,16) y[i] = x[i];

    FOR(i,20) {
        FOR(j,4) {
            FOR(m,4) t[m] = x[(5*j+4*m)%16];
```

EQUIVALENCE CHECKING

```
    tnacl        <- llvm_load_module "tweetnacl.bc";
    nacl_salsa20 <- llvm_symexec tnacl symb allocs inits results;
    nacl_as      <- abstract_symbolic nacl_salsa20;
    testAIG      <- bitblast nacl_as;
    specAIG      <- bitblast {{ \k n -> Salsa20_expansion (k,n) }}
    cec testAIG specAIG;
```

```
Proving Salsa20 equivalent between Cryptol spec and tweetnacl.
        Loading tweetnacl llvm byte code.
        Extracting the Salsa20 encryption function.
        Bit blasting the NaCl and Cryptol terms.
        Using CEC to prove equivalence.
Valid
```

ENDING NOTES

REFERENCES

- Cryptol : http://cryptol.net
- Code : https://github.com/GaloisInc/cryptol
- Slides : https://github.com/TomMD/cryptol-slides
- SAW : Initial non-commercial release Monday