Verifying Amazon's s2n Library

Joey Dodds
jdodds@galois.com

Amazon is doing good things

- Created the s2n TLS library
- Created ARG (automated reasoning group)
- Gave them power

Hired Galois!

s2n

Inspired by TLS vulnerabilities discovered by researchers

OpenSSL is 550k lines of code, with 70k dedicated to TLS

s2n is only 6k

drops some arguably insecure/less secure features

Now used for all secure Amazon Web Services (AWS) traffic

    There's a lot of that.

Galois' Job

Prove s2n correct

Start with cryptographic Algorithms

keyed-Hash Message Authentication Code (HMAC)

Deterministic Random Bit Generator (DRBG)

Cryptol: A specification language for Cryptography

We want to convince Cryptographers

Here are two specification side by side:

```
Hmac H K text =

H(K XOR opad, H(K XOR ipad, text))
opad = 0x5c5c5c...5c5c
ipad = 0x363636...3636
```

```
// H((K' xor opad) || H((K' xor ipad) || message))
hmac hash hash2 hash3 key message = hash2 (okey #
internal)
 where
  ks = kinit hash3 key // K'
  okey = [k ^ 0x5C | k <- ks] // K' xor opad
  ikey = [k ^ 0x36 | k <- ks] // K' xor ipad
  // H((K' xor ipad) || message)
  internal = split (hash (ikey # message))
```

```
// H((K' xor opad) || H((K' xor ipad) || message))
hmac hash hash2 hash3 key message = hash2 (okey #
internal)
 where
  ks = kinit hash3 key // K'
  okey = [k ^ 0x5C | k <- ks] // K' xor opad
  ikey = [k ^ 0x36 | k <- ks] // K' xor ipad
  // H((K' xor ipad) || message)
  internal = split (hash (ikey # message))
```

```
int s2n_hmac_init(struct s2n_hmac_state *state, s2n_hmac_algorithm alg, const void *key, uint32_t klen)
{
    s2n_hash_algorithm hash_alg;
    state->currently_in_hash_block = 0;

    GUARD(s2n_hmac_hash_alg(alg, &hash_alg));
    GUARD(s2n_hmac_digest_size(alg, &state->digest_size));
    GUARD(s2n_hmac_block_size(alg, &state->block_size));
    GUARD(s2n_hmac_hash_block_size(alg, &state->hash_block_size));

    gte_check(sizeof(state->xor_pad), state->block_size);
    gte_check(sizeof(state->digest_pad), state->digest_size);

    state->alg = alg;

    if (alg == S2N_HMAC_SSLv3_SHA1 || alg == S2N_HMAC_SSLv3_MD5) {
        return s2n_sslv3_mac_init(state, alg, key, klen);
    }

    GUARD(s2n_hash_init(&state->inner_just_key, hash_alg));
    GUARD(s2n_hash_init(&state->outer, hash_alg));

    uint32_t copied = klen;
    if (klen > state->block_size) {
        GUARD(s2n_hash_update(&state->outer, key, klen));
        GUARD(s2n_hash_digest(&state->outer, state->digest_pad, state->digest_size));

        memcpy_check(state->xor_pad, state->digest_pad, state->digest_size);
        copied = state->digest_size;
    } else {
        memcpy_check(state->xor_pad, key, klen);
    }

    for (int i = 0; i < copied; i++) {
        state->xor_pad[i] ^= 0x36;
    }
    for (int i = copied; i < state->block_size; i++) {
        state->xor_pad[i] = 0x36;
    }

    GUARD(s2n_hash_update(&state->inner_just_key, state->xor_pad, state->block_size));

    /* 0x36 xor 0x5c == 0x6a */
    for (int i = 0; i < state->block_size; i++) {
        state->xor_pad[i] ^= 0x6a;
    }

    return s2n_hmac_reset(state);
}

int s2n_hmac_update(struct s2n_hmac_state *state, const void *in, uint32_t size)
{
    /* Keep track of how much of the current hash block is full
     *
```

SAW

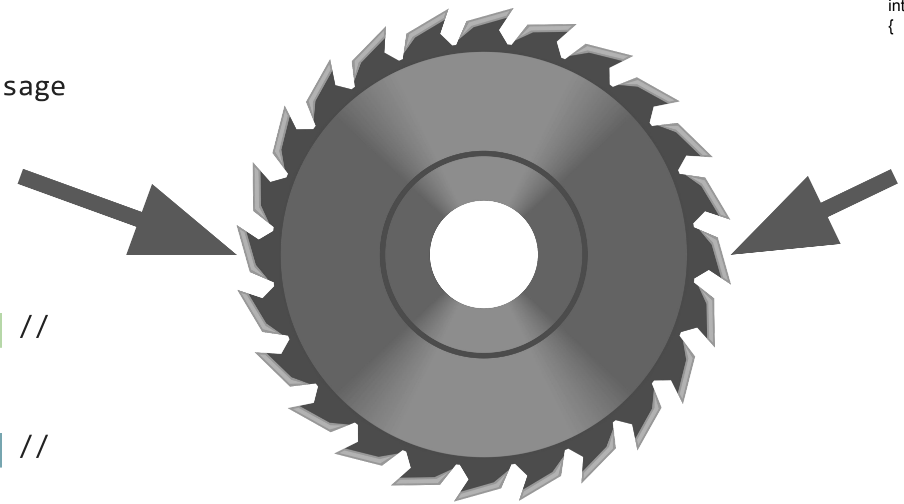
Equivalence Proofs between two programs

- Java
- Cryptol
- C (LLVM)

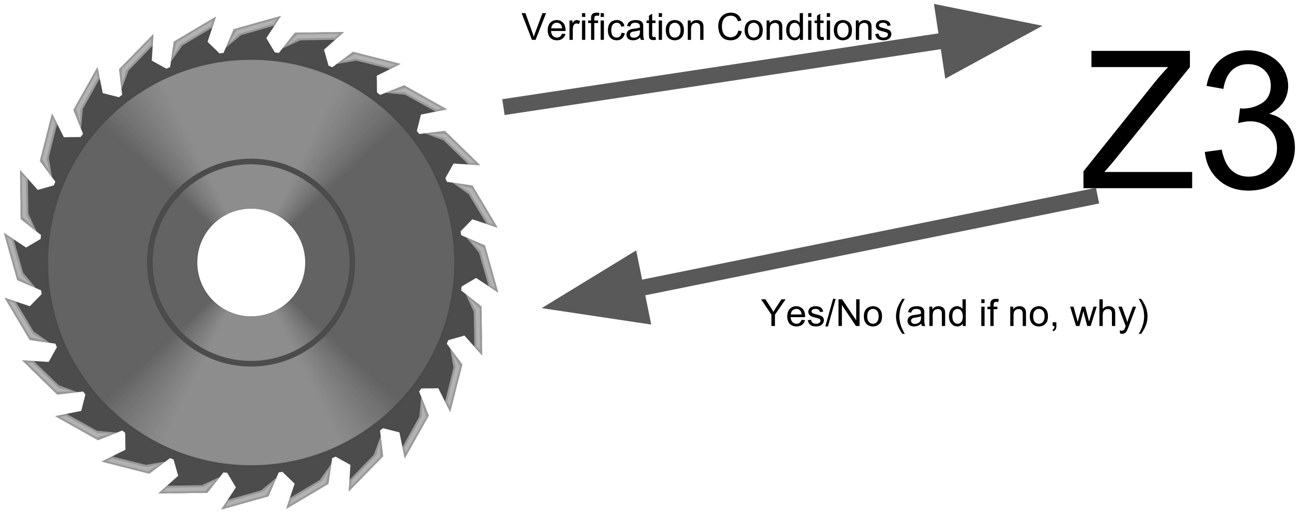
SAW-Core

Symbolic simulation of programs creates SAW-Core representations

```
opad) || H((K’ xor
sage))
sh2 hash3 key message
 # internal)

hash3 key // K’
 0x5C | k <- ks] //

 0x36 | k <- ks] //

or ipad) || message)
split (hash (ikey #
```



```
int s2n_hmac_init(struct s2n_hmac_state *state, s2n_hmac_algorithm alg, const void *key, uint32_t klen)
{
    s2n_hash_algorithm hash_alg;
    state->currently_in_hash_block = 0;

    GUARD(s2n_hmac_hash_alg(alg, &hash_alg));
    GUARD(s2n_hmac_digest_size(alg, &state->digest_size));
    GUARD(s2n_hmac_block_size(alg, &state->block_size));
    GUARD(s2n_hmac_hash_block_size(alg, &state->hash_block_size));

    gte_check(sizeof(state->xor_pad), state->block_size);
    gte_check(sizeof(state->digest_pad), state->digest_size);

    state->alg = alg;

    if (alg == S2N_HMAC_SSLv3_SHA1 || alg == S2N_HMAC_SSLv3_MD5) {
        return s2n_sslv3_mac_init(state, alg, key, klen);
    }

    GUARD(s2n_hash_init(&state->inner_just_key, hash_alg));
    GUARD(s2n_hash_init(&state->outer, hash_alg));

    uint32_t copied = klen;
    if (klen > state->block_size) {
        GUARD(s2n_hash_update(&state->outer, key, klen));
        GUARD(s2n_hash_digest(&state->outer, state->digest_pad, state->digest_size));

        memcpy_check(state->xor_pad, state->digest_pad, state->digest_size);
        copied = state->digest_size;
    } else {
        memcpy_check(state->xor_pad, key, klen);
    }
```


Verification Conditions
Z3
Yes/No (and if no, why)

Travis CI

[Here's the site]

[Here's the configuration]

Metrics Reporting

Here's the site

TLS

TLS (newer version of SSL) provides us most of the

Confidentiality

Data-Integrity

Authentication

We enjoy on the internet today

If I go to my gmail...

TLS lets me be sure I’m actually talking to google

TLS ensures that nobody (not even my ISP) can read what I’m reading*

TLS ensures that nobody (not even my ISP) can change the data I’m reading*

*Email is a terribly insecure protocol, and both of these can easily be violated when the message is sent

TLS

Some recent vulnerabilities in TLS involve bugs in the handshake protocol:

- 3SHAKE[1]
- SMACK/FREAK[2]
- Logjam[3]
- Early CCS[4]

TLS

s2n's state machine logic is

- Designed with these attacks in mind[4]
- Architected to make such bugs much less likely

Can we verify that the code achieves these goals?

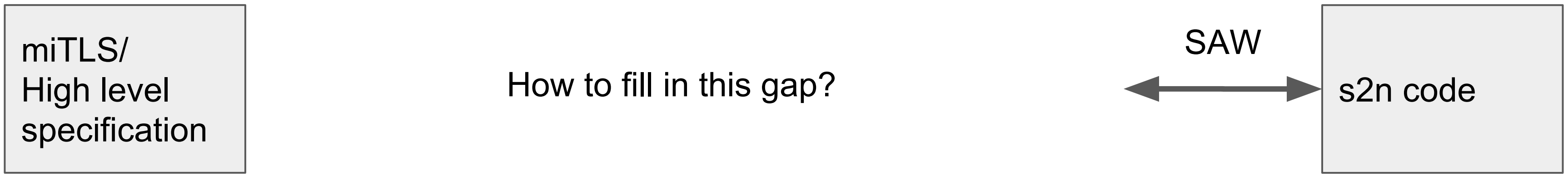
- s2n's implementation of TLS handshake permits *only* traces that are valid according to the spec

Whither specs for TLS? – Enter miTLS

TLS specification that guarantees freedom from:

- 3SHAKE[1]
- SMACK/FREAK[2]
- Logjam[3]
- Early CCS[4]

Our Approach



Extending SAW for TLS

SAW doesn't do inductive reasoning

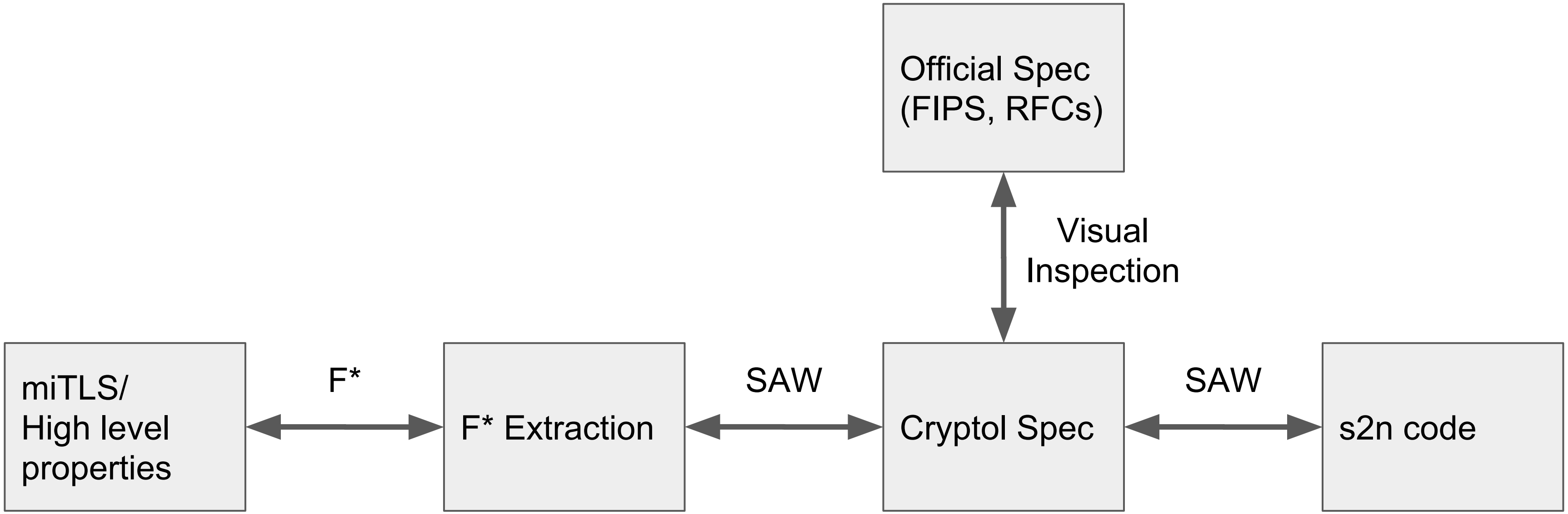
We could:

Extend SAW

Make use of a tool that is already good at inductive reasoning

Ideally: Find such a tool that has been already been used for TLS proofs

Our Approach



New Capabilities

We can now use our SAW/Cryptol specs for:

- Protocol Proof
- Cryptographic Proof
- Arbitrary Inductive Properties