Lightweight Crypto in Reverse

Dominik Táborský
dominik.taborsky@protonmail.ch

30 August 2018

Motivation

Why is reversible computing cool?

Question:

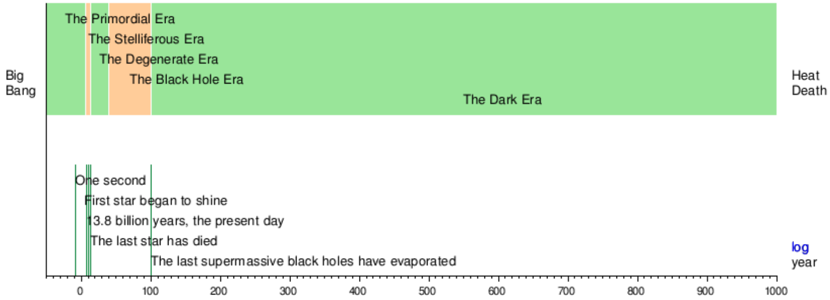
What is the universe going to look like in 10^{40} years?

Motivation

Why is reversible computing cool?

Question:

What is the universe going to look like in 10^{40} years?



Source: Wikipedia

(Proton decay is a different problem.)

Motivation

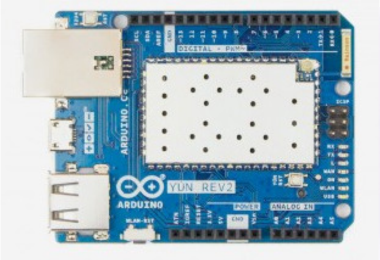
Why is lightweight cryptography cool?

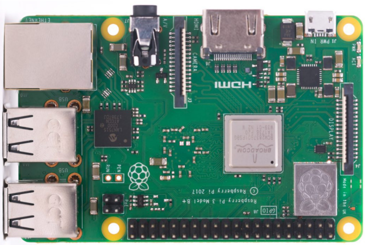
Applications

IoT and related: small systems with limited power (in any sense).

Simplicity

Mostly only symmetric cryptography, simpler and more efficient algorithms.





Source: Arduino & Raspberry Pi

Symmetric Cryptography

Symmetric Cryptography

Duality

Symmetric and crypto functions are bijective functions with a clear encryption/decryption duality.

Symmetric Cryptography

Duality

Symmetric and crypto functions are bijective functions with a clear encryption/decryption duality.

I/O

Input and output are similar with having at least two parameters: the key and the plaintext or ciphertext.

Symmetric Cryptography

Duality

Symmetric and crypto functions are bijective functions with a clear encryption/decryption duality.

I/O

Input and output are similar with having at least two parameters: the key and the plaintext or ciphertext.

$$
\begin{aligned}
[\![enc]\!](k, p) &= (k, c) \\
[\![dec]\!](k, c) &= [\![\mathcal{I}(enc)]\!](k, c) = (k, p)\,.
\end{aligned}
$$

Side channel vulnerabilities

Side channel vulnerabilities

Algorithm's mathematical strength is one part of the security, its implementation is another.

Side channel vulnerabilities

Algorithm's mathematical strength is one part of the security, its implementation is another.

Implementation can be just as hard, especially when the programmers have to actively fight the compilers.

State information leakage, timing differences, power consumption, EM radiation, noise, etc.

Goals

What are we trying to achieve?

Goals

What are we trying to achieve?

1. Protect against state information leakage.

Goals

What are we trying to achieve?

1. Protect against state information leakage.

2. Simplify implementation by exploiting reversibility with bijections.

Goals

What are we trying to achieve?

1. Protect against state information leakage.

2. Simplify implementation by exploiting reversibility with bijections.

3. Gain experience with Janus and improve it.

Goals

What are we trying to achieve?

1. Protect against state information leakage.

2. Simplify implementation by exploiting reversibility with bijections.

3. Gain experience with Janus and improve it.

Have we reached them?

Goals

What are we trying to achieve?

1. Protect against state information leakage.

2. Simplify implementation by exploiting reversibility with bijections.

3. Gain experience with Janus and improve it.

Have we reached them? Yes!

Avoiding state leakage

Goal #1

Avoiding state leakage

Goal #1

Reversible code cannot erase information.

- Information erasure would prevent backward determinism.
- Memory cannot be reset and is asumed to be zero.
- This enforces proper memory cleanup to avoid memory leakage.

Avoiding state leakage

Goal #1

Reversible code cannot erase information.

- Information erasure would prevent backward determinism.
- Memory cannot be reset and is asumed to be zero.
- This enforces proper memory cleanup to avoid memory leakage.

Whatever information we start with, the same information we also end up with, only transformed.

Avoiding state leakage

Goal #1

Reversible code cannot erase information.

- Information erasure would prevent backward determinism.
- Memory cannot be reset and is asumed to be zero.
- This enforces proper memory cleanup to avoid memory leakage.

Whatever information we start with, the same information we also end up with, only transformed.
The computational state information is only temporary, stored within local variables.

Example

Goal #1

```
procedure TEA_encipher(u32 data[], u32 key[])
    local u32 delta = 2654435769
    local u32 sum = 0
    iterate int i = 1 by 1 to 32
        sum += delta
        data[0] += ((data[1] * 16) + key[0]) ^
                   (data[1] + sum) ^
                   ((data[1] / 32) + key[1])
        data[1] += ((data[0] * 16) + key[2]) ^
                   (data[0] + sum) ^
                   ((data[0] / 32) + key[3])
    end
    delocal u32 sum = 32*delta
    delocal u32 delta =2654435769
```

Simplifying programming work

Goal #2

Simplifying programming work

Goal #2

Encryption is bijection

Implementing just encryption (or just decryption) is sufficient. The other function is simply its interpretation in reverse.

Simplifying programming work

Goal #2

Encryption is bijection

Implementing just encryption (or just decryption) is sufficient. The other function is simply its interpretation in reverse.

Saves time on:

1. Writing the inverse function code
2. Testing for its correctness and its inverseness to the other function
3. Debugging

Example

Goal #2

```
show(data)
call TEA_encipher(data, key)
show(data)
uncall TEA_encipher(data, key)
show(data)
```

```
data[2] = {42, 27}
data[2] = {1535266570, 1744185122}
data[2] = {42, 27}
```

Learning and improving Janus

Goal #3

Learning and improving Janus

Goal #3

Learning

- Implemented several crypto algorithms in Janus
- Re-discovered both Bennett's tricks

Learning and improving Janus

Goal #3

Learning

- Implemented several crypto algorithms in Janus
- Re-discovered both Bennett's tricks

Improving

- Fixed bugs in Jana
- Made improvements to Jana
- Suggested two Janus extensions

Example

Goal #3

```
// z1 = y1 ^ ((y0 +y3) <<< 7)
tmp += seq[0] + seq[3]
call rotate_left_u32(tmp, 7)
seq[1] ^= tmp
uncall rotate_left_u32(tmp, 7)
tmp -= (seq[0] + seq[3])

// z1 = y1 ^ ((y0 +y3) <<< 7)
seq[1] ^= call rotate_left_u32(seq[0] +seq[3], 7)
```

Experiment

Evaluating the first goal

Experiment

Evaluating the first goal

1. Examining leakage in both reference and our implementations
2. Evaluating alternatives

Experiment

Evaluating the first goal

1. Examining leakage in both reference and our implementations
2. Evaluating alternatives

1. Secretgrind
2. Vale, zerostack

Examining leakage

```
***(1) (stack)   range [0xffefffd40 - 0xffefffdff]
    (192 bytes) is tainted
Total bytes tainted: 192

***(1) (stack)   range [0xffefffdc0 - 0xffefffdff] (64
     bytes) is tainted
   > (stack) [0xffefffdc0 - 0xffefffdc3] (4 bytes):
      Chacha20.janus.3.cpp:702:@0xffefffdc0:seq
        ...
   > (stack) [0xffefffdc4 - 0xffefffdc7] (4 bytes):
      Chacha20.janus.3.cpp:702:@0xffefffdc4:seq[1]
        ...
   > (stack) [0xffefffdc8 - 0xffefffdff] (56 bytes):
      Chacha20.janus.3.cpp:702:@0xffefffdc0:seq
        ...
Total bytes tainted: 64
```

Evaluating alternatives

Evaluating alternatives

Vale

- Not mature enough
- Problematic to set up
- Very broad to understand
- Difficult to use
- Limited documentation
- Huge potential

Evaluating alternatives

Vale

- Not mature enough
- Problematic to set up
- Very broad to understand
- Difficult to use
- Limited documentation
- Huge potential

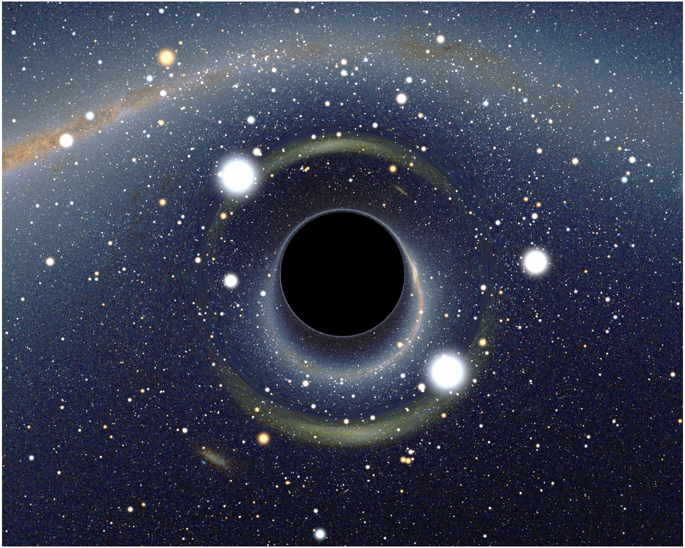
zerostack

- Much easier to set up
- Extremely simple to understand and use
- Documentation unnecessary due to simplicity
- Not mature (supports only single configuration)
- Limited potential

Contributions

1. Shown usefulness of reversibility in cryptography
2. Implemented several cryptographic algorithms
3. Improved Janus and Jana
4. Research published in LNCS
5. All work available on GitHub

Remember the black holes!



Source: Wikipedia

That's it!

Thank you!
Questions?