# Reveal Secrets in Adoring Poitras

## A victory of reverse engineering and cryptanalysis over challenge 777

Louis Goubin[1,4] Pascal Paillier[1]
Matthieu Rivain[1] Junwei Wang[1,2,3]

[1]CryptoExperts

[2]University of Luxembourg

[3]University of Paris 8

[4]University of Versailles-St-Quentin-en-Yvelines

CHES 2017, Rump Session, Taipei

CRYPTOEXPERTS

UNIVERSITÉ DU LUXEMBOURG

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

ECRYPT NET

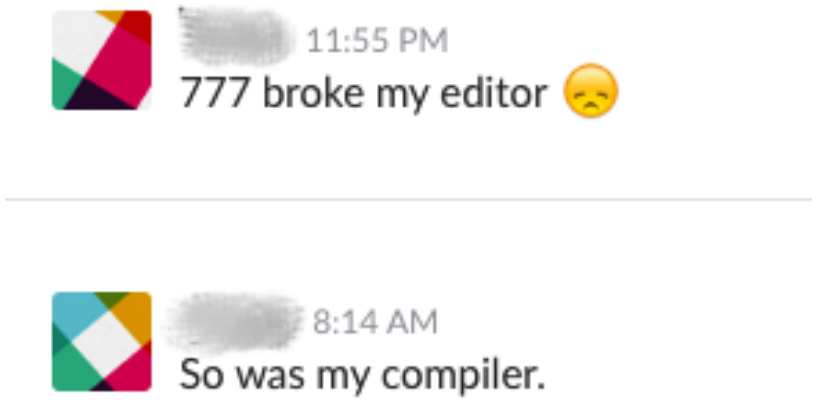
# Outline

0. Downloading and Compiling the Code
1. Cleaning the Code
2. De-Virtualization
3. From Bitwise Program to Boolean Circuits
4. Boolean Circuits Minimization
5. Data Dependency Analysis
6. Algebraic Analysis

# Outline

# Downloading and Compiling the code Code

- Browsers stuck at loading it...
- Editors are broken by it...
- Some compilers (e.g., llvm) keep compiling and reporting warnings...



# Outline

# Untidy Code

## More than 1k functions

# Readability Processing

- Duplicate / redundancy / unused codes elimination
- Functions / variables renaming
- Constants rewriting
- Code combination

# Readability Processing

- Duplicate / redundancy / unused codes elimination
- Functions / variables renaming
- Constants rewriting
- Code combination

Only 20 functions are remaining

```
void copy(uint KLedyCW, uint lRjmjY) {int_arr[(a+KLedyCW) & 0x3ffff] = int_arr[lRjmjY & 0x3ffff];}
void encode(uint owhjO, uint nlBqXn) {assign(owhjO, in_ptr[nlBqXn]);}
void decode(uint hFqeIO, uint jvXpt) {out_ptr[hFqeIO]=int_arr[(a+jvXpt)&0x3ffff];}
void rshift_xor(uint HcCOL, uint ISRfdIp, uint ufYFMX) { int_arr[HcCOL&0x3ffff]^=1&(int_arr[(a+ISRfdIp)&0x3ffff]>>ufYFMX); /*prir
void lshift_xor(uint NcJBw, uint OMQBXqa, uint AyoLFz) { uint dyfS=(int_arr[(a+AyoLFz)&0x3ffff])&1;int_arr[(a+NcJBw)&0x3ffff]^=dy

void expand_bit(uint SteQlD, uint ZubEP, uint ZiQz) {int_arr[(a+SteQlD)&0x3ffff]=-((int_arr[(a+ZubEP)&0x3ffff]>>ZiQz)&1);}

uint lookup1(uint AKBKig) {return int_arr[(a+AKBKig)&0x3ffff];}
uint lookup2(uint WAdV, uint ZcVdJ) {return int_arr[(WAdV+ZcVdJ)&0x3ffff];}

void assign(uint UbEJi, uint UmwjUhO) {int_arr[(a+UbEJi)&0x3ffff]=UmwjUhO;}
void assign_a(uint wEOkx) {a = wEOkx;}
void assign_b(uint fnmqxL) {b=int_arr[fnmqxL]&0x07fff;}
void update_a() {a=lookup2(1592,mix(b)); printf("%lu\n",a);}
void update_b() {b=0x7fff&lookup2(522,mix(b));}

void mistery(uint wJxea, uint QBGXUN) {uint t = (~int_arr[(a+QBGXUN)&0x3ffff])&0x7fff; assign(wJxea,lookup2(2979,mix(t)));}

// bitwise operation
void xor(uint oEHmwk, uint KCZu, uint MtCA) {int_arr[(a+oEHmwk)&0x3ffff]=int_arr[(a+KCZu)&0x3ffff]^int_arr[(a+MtCA)&0x3ffff];}
void and(uint bmmFp, uint UNFg, uint PqCtZYZ) {uint t = int_arr[(a+UNFg)&0x3ffff]&int_arr[(a+PqCtZYZ)&0x3ffff];}
void or(uint eTGI, uint udoXFs, uint mezPNN) {int_arr[(a+eTGI)&0x3ffff]=int_arr[(a+udoXFs)&0x3ffff]|int_arr[(a+mezPNN)&0x3ffff];}
void not(uint YfnT, uint JKTw) {int_arr[(a+YfnT)&0x3ffff]=~int_arr[(a+JKTw)&0x3ffff];}

// jump
void goto_f(uint LKhOC) {pc = bop + LKhOC;}
void jump_if(uint DbvJO, uint FleFNIf, uint LeHf) { if(lookup2(2979,mix(b))==lookup2(DbvJO, FleFNIf) || count >= 64) {printf("%d
```

# Outline

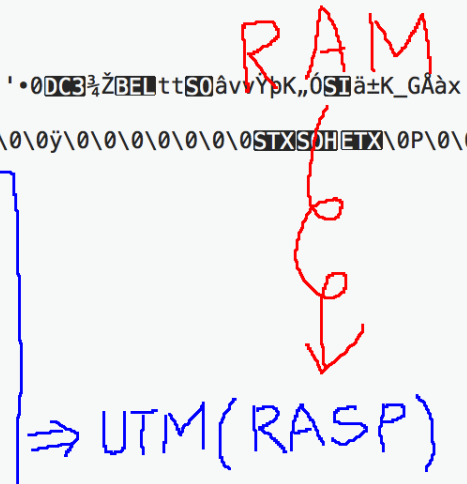
# Universal Turing Machine

```
2262 uint kIKfgI, VDpRwo;
2263 uchar *JqcadL;
2264 uchar *xkpRp, *Puix;
2265 uchar *sutlnu;
2266 uchar *unYAQ;
2267 const char pDeoW[] = "äSTXUS}BS&ýÖ:eK±Ï¿üì¡÷¿žåPê‹˜Ùþç•Àâ]'•0DC3¾ŽBELttSOâvvŸþK„ÓSIä±K_GÅàx    ETB¿íf
2268 uint ooGoRv[sizeof(pDeoW)/sizeof(uint)];
2269 char JGNNvi[] = "SOH\0\n\0\0\0\0\0\0STXSOHxSUBSOH\0\0\0\0\0ÿ\0\0\0\0\0\0\0STXSOHETX\0P\0\0\0\0\0\0
2270 uchar *klspCVy = (uchar*)JGNNvi;
2271 void aLLl() {
2272 unYAQ = klspCVy;
2273 sutlnu = klspCVy + sizeof(JGNNvi)/sizeof(uchar);
2274 JqcadL = klspCVy;
2275 while (JqcadL < sutlnu) {
2276 uchar eMmr = *JqcadL++;
2277 if (eMmr == 0) {
2278 void (*QiEb)();
2279 QiEb = (void*)funcptrs[*JqcadL++];
2280 uint *AnezsV = (uint*)JqcadL;
2281 JqcadL += eMmr*8;
2282 QiEb();
2283 }
2284 else if (eMmr == 1) {
2285 void (*QiEb)(uint);
2286 QiEb = (void*)funcptrs[*JqcadL++];
2287 uint *AnezsV = (uint*)JqcadL;
2288 JqcadL += eMmr*8;
2289 QiEb(AnezsV[0]);
2290 }
2291 ...
2327 }
2328 void AES_128_encrypt(uchar *OLjd, uchar *xzptiF) {
2329 for(kIKfgI = 0; kIKfgI < sizeof(ooGoRv)/sizeof(uint); kIKfgI++) ooGoRv[kIKfgI] = gBXW[kIKfgI];
2330 xkpRp = OLjd;
2331 Puix = xzptiF;
2332 if (sizeof(klspCVy))
2333 aLLl();
2334 else
2335 1;// c;
2336 }
```

⇒ UTM(RASP)

# Universal Turing Machine (2)

```
2262 uint kIKfgI, VDpRwo;
2263 uchar *JqcadL;
2264 uchar *xkpRp, *Puix;
2265 uchar *sutlnu;
2266 uchar *unYAQ;
2267 const char pDeoW[] = "äSTXUS}BS&ýÖ:eK±Ï¿üì¡÷¿žåPê‹˜Ùþç•Àâ]'•0DC3¾ŽBELttSOâvvÝÞK„ÓSIä±K_GÅàx    ETB¿íf
2268 uint ooGoRv[sizeof(pDeoW)/sizeof(uint)];
2269 char JGNNvi[] = "SOH\0\n\0\0\0\0\0\0STXSOHxSUBSOH\0\0\0\0\0ÿ\0\0\0\0\0\0\0STXSOHETX\0P\0\0\0\0\0\0
2270 uchar *klspCVy = (uchar*)JGNNvi;
2271 void aLLl() {
2272 unYAQ = klspCVy;
2273 sutlnu = klspCVy + sizeof(JGNNvi)/sizeof(uchar);
2274 JqcadL = klspCVy;
2275 while (JqcadL < sutlnu) {
2276 uchar eMmr = *JqcadL++;
2277 if (eMmr == 0) {
2278 void (*QiEb)();
2279 QiEb = (void*)funcptrs[*JqcadL++];
2280 uint *AnezsV = (uint*)JqcadL;
2281 JqcadL += eMmr*8;
2282 QiEb();
2283 }
2284 else if (eMmr == 1) {
2285 void (*QiEb)(uint);
2286 QiEb = (void*)funcptrs[*JqcadL++];
2287 uint *AnezsV = (uint*)JqcadL;
2288 JqcadL += eMmr*8;
2289 QiEb(AnezsV[0]);
2290 }
2291 ...
2327 }
2328 void AES_128_encrypt(uchar *OLjd, uchar *xzptiF) {
2329 for(kIKfgI = 0; kIKfgI < sizeof(ooGoRv)/sizeof(uint); kIKfgI++) ooGoRv[kIKfgI] = gBXW[kIKfgI];
2330 xkpRp = OLjd;
2331 Puix = xzptiF;
2332 if (sizeof(klspCVy))
2333 aLLl();
2334 else
2335 1;// c;
2336 }
```



# Universal Turing Machine (3)

```
2262 uint kIKfgI, VDpRwo;
2263 uchar *JqcadL;
2264 uchar *xkpRp, *Puix;
2265 uchar *sutlnu;
2266 uchar *unYAQ;
2267 const char pDeoW[] = "äSTXUS}BS&ýÖ:eK±Ï¿üìì÷¿žåPê‹˜Ùþç•Àâ]'•0DC3¾ŽBELttS0âvvYÞK„ÓSTä±K_GÂàx    ETB¿íf
2268 uint ooGoRv[sizeof(pDeoW)/sizeof(uint)];
2269 char JGNNvi[] = "SOH\0\n\0\0\0\0\0\0\0STXSOHxSUBSOH\0\0\0\0\0ÿ\0\0\0\0\0\0\0STXSOHETX\0P\0\0\0\0\0\0
2270 uchar *klspCVy = (uchar*)JGNNvi;
2271 void aLLl() {
2272 unYAQ = klspCVy;
2273 sutlnu = klspCVy + sizeof(JGNNvi)/sizeof(uchar);
2274 JqcadL = klspCVy;
2275 while (JqcadL < sutlnu) {
2276 uchar eMmr = *JqcadL++;
2277 if (eMmr == 0) {
2278 void (*QiEb)();
2279 QiEb = (void*)funcptrs[*JqcadL++];
2280 uint *AnezsV = (uint*)JqcadL;
2281 JqcadL += eMmr*8;
2282 QiEb();
2283 }
2284 else if (eMmr == 1) {
2285 void (*QiEb)(uint);
2286 QiEb = (void*)funcptrs[*JqcadL++];
2287 uint *AnezsV = (uint*)JqcadL;
2288 JqcadL += eMmr*8;
2289 QiEb(AnezsV[0]);
2290 }
2291 ...
2327 }
2328 void AES_128_encrypt(uchar *OLjd, uchar *xzptiF) {
2329 for(kIKfgI = 0; kIKfgI < sizeof(ooGoRv)/sizeof(uint); kIKfgI++) ooGoRv[kIKfgI] = gBXW[kIKfgI];
2330 xkpRp = OLjd;
2331 Puix = xzptiF;
2332 if (sizeof(klspCVy))
2333 aLLl();
2334 else
2335 1;// c;
2336 }
```

# De-virtualization - Simulate the UTM

```
    else if (eMmr == 3) {
        void (*QiEb)(uint, uint, uint);
        QiEb = (void*)funcptrs[*pc++];
        uint *AnezsV = (uint*)pc;
        pc += eMmr*8;
        // QiEb(AnezsV[0], AnezsV[1], AnezsV[2]);
#ifdef SIMULATE
        printf("%8s(%d,%d,%d);\n",flist[*(pc-1-eMmr*8)],AnezsV[0],AnezsV[1],AnezsV[2]);
#endif
    }
    else if (eMmr == 4) {
        void (*QiEb)(uint, uint, uint, uint);
        QiEb = (void*)funcptrs[*pc++];
        uint *AnezsV = (uint*)pc;
        pc += eMmr*8;
        // QiEb(AnezsV[0], AnezsV[1], AnezsV[2], AnezsV[3]);
#ifdef SIMULATE
        printf("%8s(%d,%d,%d,%d);\n",flist[*(pc-1-eMmr*8)],AnezsV[0],AnezsV[1],AnezsV[2],AnezsV[3]);
#endif
    }
```

# De-virtualization - Simulate the UTM

```
    else if (eMmr == 3) {
        void (*QiEb)(uint, uint, uint);
        QiEb = (void*)funcptrs[*pc++];
        uint *AnezsV = (uint*)pc;
        pc += eMmr*8;
        // QiEb(AnezsV[0], AnezsV[1], AnezsV[2]);
#ifdef SIMULATE
        printf("%8s(%d,%d,%d);\n",flist[*(pc-1-eMmr*8)],AnezsV[0],AnezsV[1],AnezsV[2]);
#endif
    }
    else if (eMmr == 4) {
        void (*QiEb)(uint, uint, uint, uint);
        QiEb = (void*)funcptrs[*pc++];
        uint *AnezsV = (uint*)pc;
        pc += eMmr*8;
        // QiEb(AnezsV[0], AnezsV[1], AnezsV[2], AnezsV[3]);
#ifdef SIMULATE
        printf("%8s(%d,%d,%d,%d);\n",flist[*(pc-1-eMmr*8)],AnezsV[0],AnezsV[1],AnezsV[2],AnezsV[3]);
#endif
    }
```

We get a bitwise-based program (600k operations).

# Outline

# Bitwise-based program

**Input:** plaintext bits $(b_1, b_2, \cdots, b_{128})$
**Output:** ciphertext bits $(c_1, c_2, \cdots, c_{128})$

**for** $i = 1$ to 128 **do**
  $t[addr_{1,i}] \leftarrow 0\mathrm{b}b_i b_i b_i \cdots b_i$ ▷ expand $b_i$ to unsigned long integer (64 bits)
  **for** $j = 1$ to 64 **do**
    $t[addr_{2,i} + j * 2^{12}] \leftarrow t[addr_{1,i}]$
  **end for**
**end for**

BitwiseOperationLoop1 ▷ loop for 64 times
BitwiseOperationLoop2
...
BitwiseOperationLoop2573

**for** $i = 1$ to 129 **do**
  $t[addr_{3,i}] \leftarrow v_i$ ▷ $v_i \in \mathtt{GF}(2)$ is a constant
  **for** $j = 1$ to 64 **do**
    $tmp \leftarrow t[addr_{4,i} + j * 2^{12}] \oplus t[addr_{5,i} + j * 2^{12}]$
    $t[addr_{3,i}] \leftarrow t[addr_{3,i}] \oplus$ Parity$(tmp)$ ▷ Parity computes the number of 1-bit modulo 2
  **end for**
**end for**

BitwiseOperationLoop2574
...
BitwiseOperationLoop2582

**for** $i = 1$ to 128 **do**
  $c_i \leftarrow t[addr_{6,i}]$
**end for**

# From Bitwise Program to Boolean Circuits

- 64 (loop length) * 64 (number of bits in a unsigned long integer) independent AES computations operated in boolean circuits
- 3 out of 64*64 are the real and identical AES computations (e.g., bit 42 of loop 26)
- Hence, the bitwsie-based program can be simplified as a boolean circuits with 600k gates (XOR, AND, OR, NOT).

# Outline

# Boolean Circuits Minimization

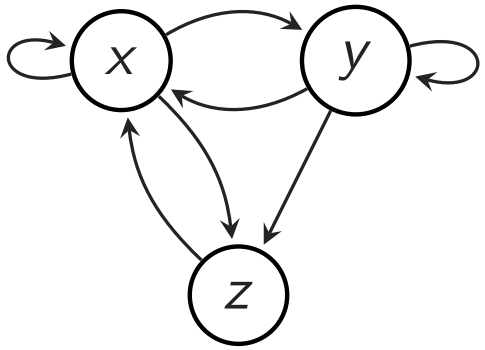
- Constant variable detection and propagation
- Dead code elimination
- Deduplication
- "Potential" pseudorandomness detection and removal
- Repeat the above steps until no more constant / duplicate / "potential" pseudorandomness can be detected

Finally, the circuits is reduced to 280k boolean gates (53% smaller)

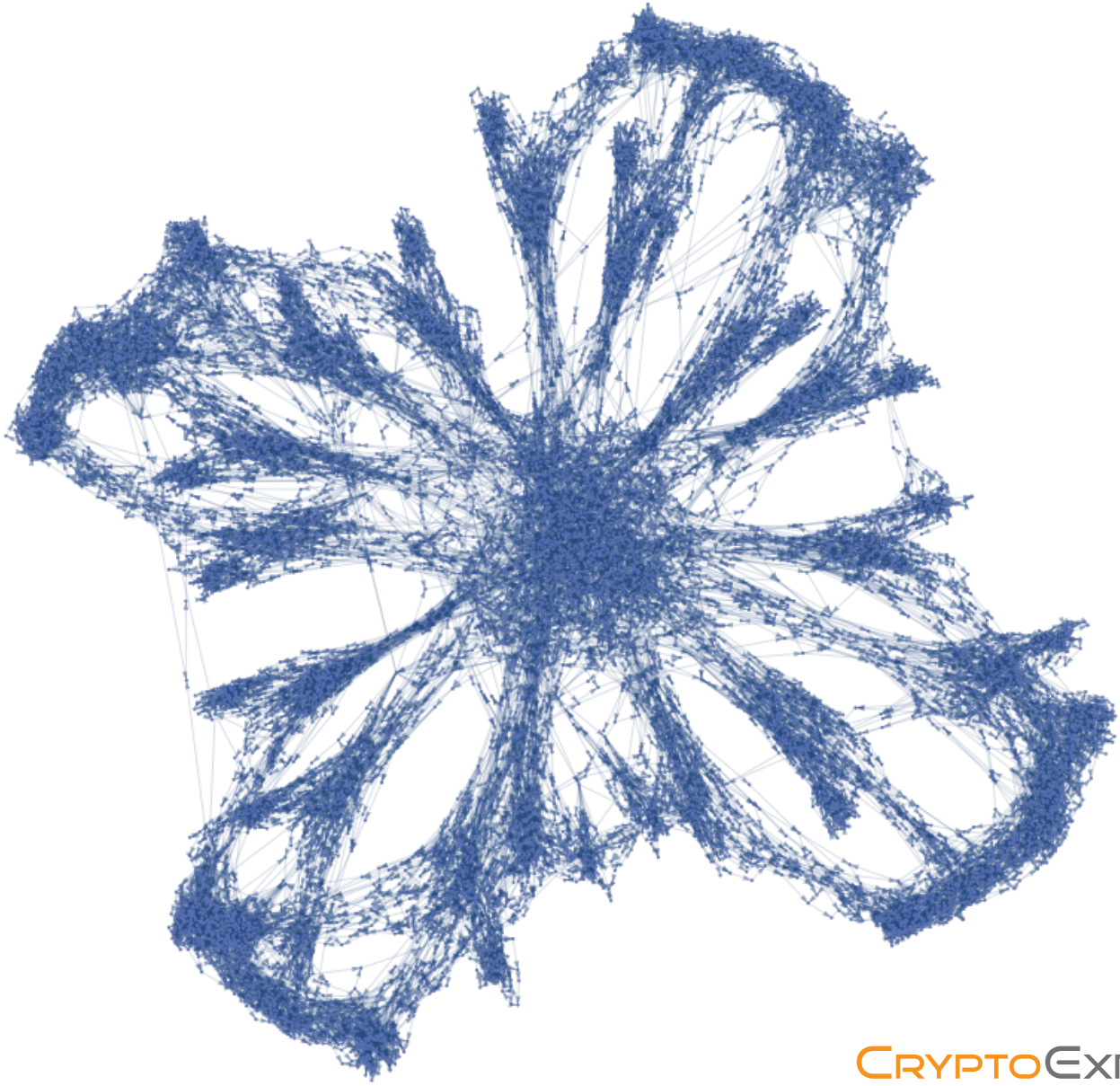
# Outline

# Data Dependency Graph (DDG)

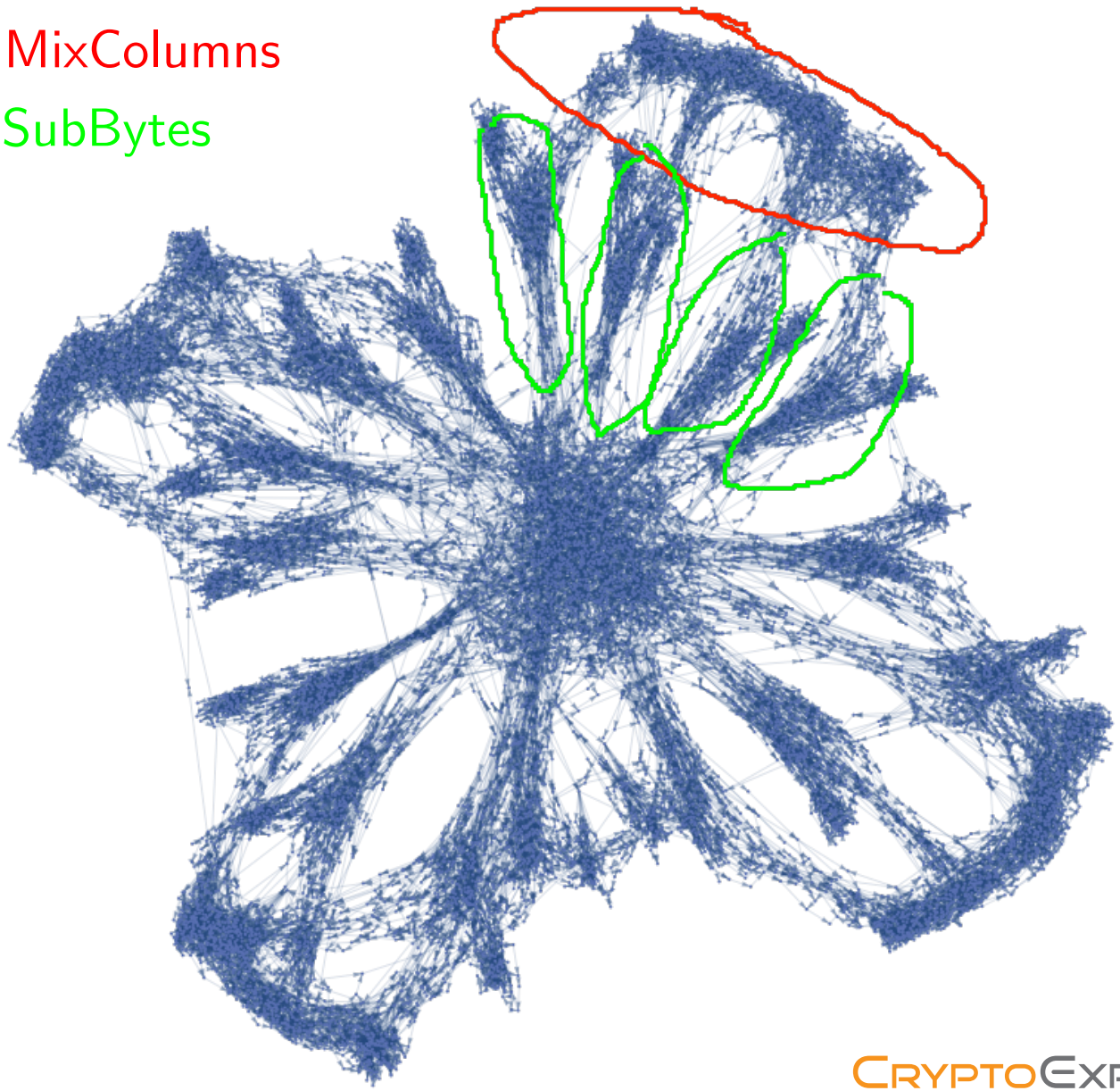
$$x = a;$$
$$y = b;$$
$$x = y + x;$$
$$y = x * y;$$
$$z = x - y;$$
$$x = z * x;$$



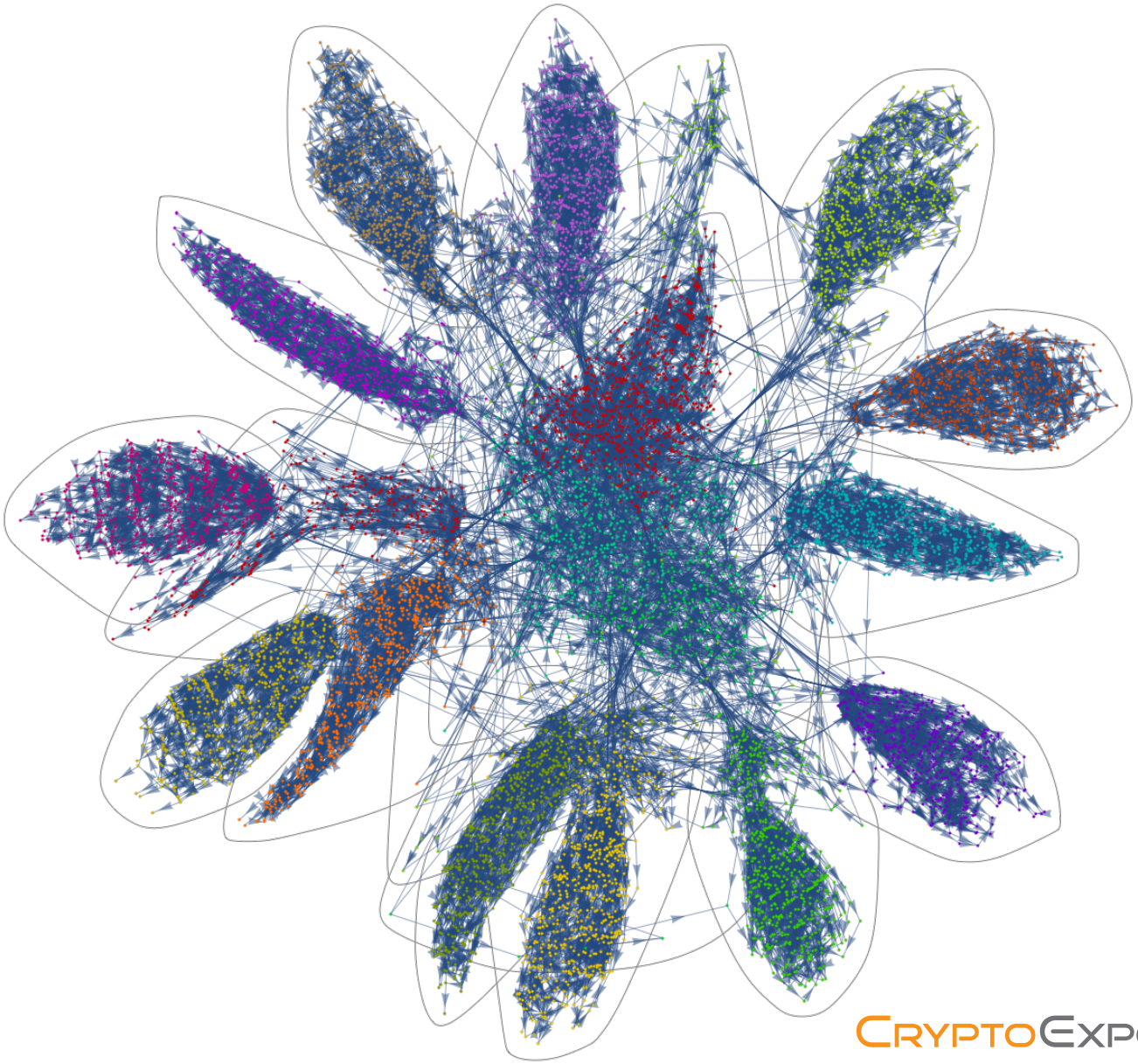
# DDG of the Circuits (First 5%)



# First Round Computation of AES



# Extracting the Branches (Clustering)



# Outline

# Assumption

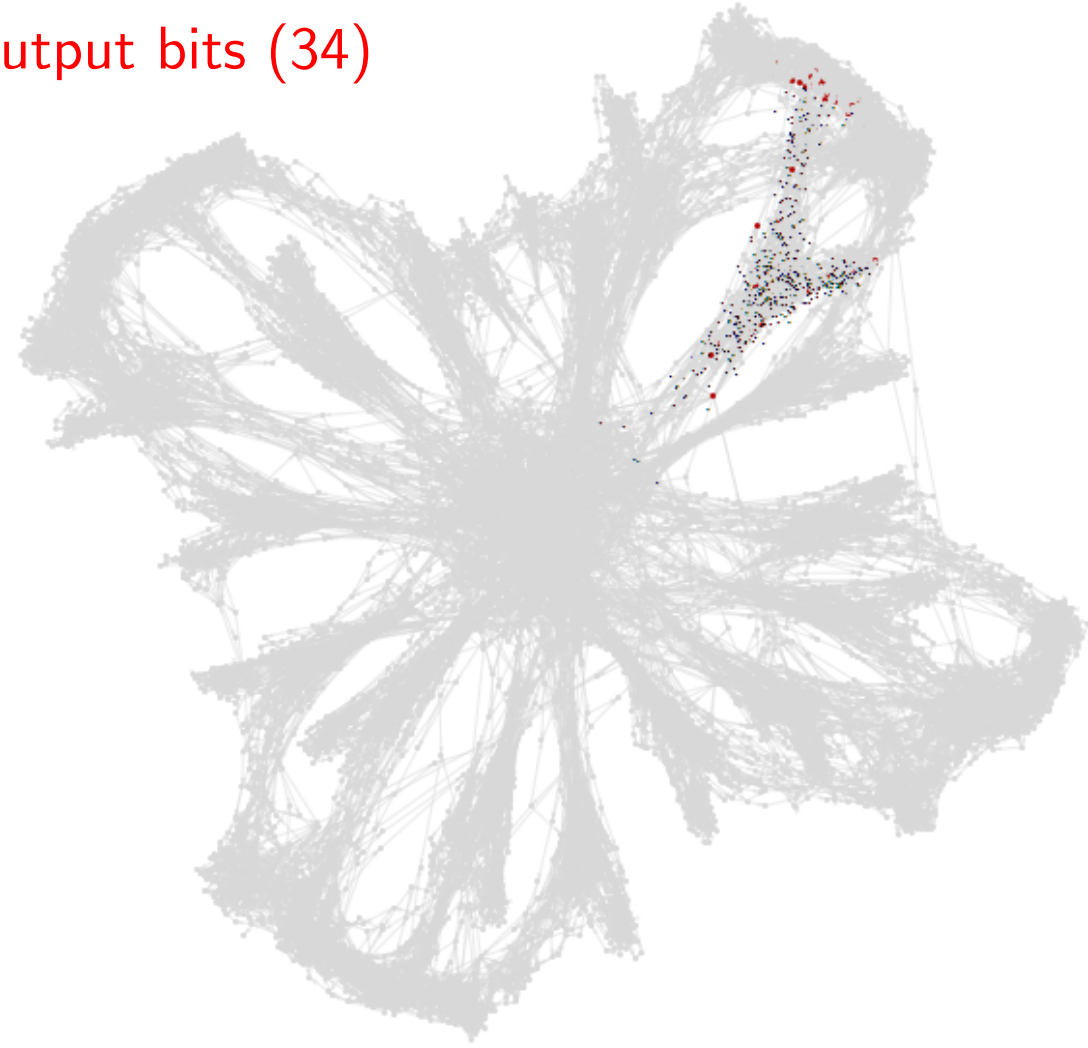
**Assumption (Informal)**

Each of the green "branch" corresponds to an individual S-Box computation in the first round of AES, the $t$-bit output $(s_1, s_2, \cdots, s_t)$ of which is a linear encoding of a real S-Box output bit.

# Output Bits of A Branch

Bits in a branch (530)

S-Box output bits (34)



# Solve a Systems of Linear Equations

$$\begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \dots & s_{34}^{(1)}, 1 \\ s_1^{(2)} & s_2^{(2)} & \dots & s_{34}^{(2)}, 1 \\ \vdots & \vdots & \ddots & \vdots \\ s_1^{(n)} & s_2^{(n)} & \dots & s_{34}^{(n)}, 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{34} \\ a_{35} \end{bmatrix} = \begin{bmatrix} \mathtt{SBox}(x^{(1)} \oplus \hat{k})[i] \\ \mathtt{SBox}(x^{(2)} \oplus \hat{k})[i] \\ \vdots \\ \mathtt{SBox}(x^{(n)} \oplus \hat{k})[i] \end{bmatrix}$$

If $n \geq 35 + 8 + \lambda$, $\Pr[\text{“}\hat{k} \neq k^* \text{ has a solution”}] \leq 2^{-\lambda}$.

# Results

```
In[488]:= LinearBreak[data]

key=0x0
key=0x10
key=0x20
key=0x30
key=0x40
key=0x50
key=0x60
key=0x70
key=0x80
key=0x90
key=0xa0
key=0xb0
key=0xc0
!!!!!!!!!!!!! 2 - 0 - 0xcf !!!!!!!!!!!!!
!!!!!!!!!!!!! 2 - 1 - 0xcf !!!!!!!!!!!!!
!!!!!!!!!!!!! 2 - 2 - 0xcf !!!!!!!!!!!!!
!!!!!!!!!!!!! 2 - 3 - 0xcf !!!!!!!!!!!!!
!!!!!!!!!!!!! 2 - 4 - 0xcf !!!!!!!!!!!!!
!!!!!!!!!!!!! 2 - 5 - 0xcf !!!!!!!!!!!!!
!!!!!!!!!!!!! 2 - 6 - 0xcf !!!!!!!!!!!!!
!!!!!!!!!!!!! 2 - 7 - 0xcf !!!!!!!!!!!!!
key=0xd0
key=0xe0
key=0xf0
```

# Thank you!