

Reveal Secrets in Adoring Poitras

A victory of reverse engineering and cryptanalysis over challenge 777

Louis Goubin^{1,4} Pascal Paillier¹
Matthieu Rivain¹ Junwei Wang^{1,2,3}

¹CryptoExperts

²University of Luxembourg

³University of Paris 8

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

CHES 2017, Rump Session, Taipei

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

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

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...

 11:55 PM
777 broke my editor 😞

 8:14 AM
So was my compiler.

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

Untidy Code

More than 1k functions

```
void x5nEq(uint UMNvLp, uint KtFY, uint vzJZq) {if(nIlajqq)==IFWBUN(UMNsVlp, KtFY)) Ewwon(vzJZq);}
void qGrcRsk() {kIKfgI=bPcrtK());}
void mCazhi(uint EwREsc, uint CLoJ) {if(Brza)==v~=
oid SNGxNy(uint lntfPv, uint iLRoIt, uint KTfY)
void rNUIPyd(uint hPqeIo, uint jvxpt) {xpkn
void uPdL(uint QRFdI, uint CoCI, uint aLPxx)
void UfJtwb(uint HCCt, uint ISRFdIp, uint urrxX) {sconvl((kIKfgI<18)&~kIKfgI>262143)^=ooGoRv[(kIKfgI+ISRFdIp)&262143]>>ufYFM
uint dLT(uint RouDc, uint TSCaTl) {return ooGoRv[763216u]|qscwtK(RouDUC+(kIKfgI<17),TSCaTl);}
void WnCwUs(uint AkUlfh, uint xtva) {xkpRp(AkUlfh)=xkUluIdu(xtva);}
uint ckn(uint ZxWn, uint dywdmc) {return (ZxWn+dywdmc)|196983;}
uint mgxgNy(uint cdq, uint aLEjS) {return TzXsSE(cdq+(kIKfgI>18),aLEjS);}
void gtdkbbX(uint OIModr, uint TuqiqX) {xkpRp(OIModr)=NuqTr(TuqiqX);}
void TbsWl(uint WTApxRg, uint Wnx) {VdpRwo=ooGoRv[MYarFxG]&Wnx;}
void puBLD(uint XBzm, uint aZGHRL) {xpjOT(XBzm, PuixAzGHRL);}
uint Mpfn(uint ghpcTsG, uint UeGvxVS) {return npLEFT(ghpcTsG+(kIKfgI<18),UeGvxVS);}
void zsBuL(uint NCJW, uint OMObXq, uint AyolFz, uint BwxtvX) {uint dyfs=(ooGoRv[(kIKfgI+AyoFz)&262143]>BwxtvX)&1;ooGoRv[(kIKfgI
void CNUL(uint UtDajEs, uint VktOvZ) {ooGoRv[(kIKfgI+UtDajEs)|113092]>>ooGoRv[(kIKfgI+VktOvZ)&8473|1]
void ZAPuRa(uint manfh, uint Gziw) {ooGoRv[(kIKfgI+manfh)&28149]=ooGoRv[((ooGoRv[(kIKfgI+Gziw)|142194])|16981)<<1]+437+(((eoYLqcs(uint GaULgs, uint xVRz, uint bTuAfhw) {if(jQdGL)==Mpfn(GaULgs, xVRz) Ioez(bTuAfhw);}
void ghalu(uint EfrHz, uint HmaOvn, uint xAxOdt) {ooGoRv[(kIKfgI+EfrHz)&262143]=(~ooGoRv[(kIKfgI+HmaOvn)|262143]>>57)&(ooGoRv[(void bqlseRp(uint Ktkq, uint ggaA5, uint aPrcDcv) {ooGoRv[(kIKfgI+ktkq)|167828]=ooGoRv[(kIKfgI+al
void veUmt(uint gomv, uint spuKo) {return ooGoRv[(onyGwJpVxO|262143);}
void XCHfHe(uint IXxBG, uint BlPTTU) {VdpRwo=ooGoRv[IXxBG]&BlPTTU];
void HKyPrf(uint ZmDf, uint udnfPv, uint JtKw) {ooGoRv[(kIKfgI+JtKw)|262143];
void dnEmK(uint WAdv, uint ZcvD) {return ooGoRv[(WAdv|ZcvD)&262143];}
void QlIx0(uint KledyCw, uint LrjmjY) {luDBBCn(KledyCw, xkUluIdu(LrjmjY-kIKfgI));}
void koPlBY(uint wamXceM) {kIKfgI=wamXceM;}
void BndSNHO(uint efusN, uint qSia) {ooGoRv[(kIKfgI+efusN)|211401]=~ooGoRv[(kIKfgI+qSia)^129545]<<25;}
void zuWrmHS(uint MnjJkN, uint LzNw) {ooGoRv[(kIKfgI+MnjJkN)|230698]=~ooGoRv[(kIKfgI+LzNw)&6145|1>>9];
void TUnc(uint DqPwv, uint chNeV, uint xtFa) {ooGoRv[(kIKfgI+DqPwv)&262143]=~ooGoRv[(kIKfgI+chNeV)&262143]<<23)+~ooGoRv[(kIKfgI
Euc(uint vrl, uint vise) {VdpRwo=0hrPwv(InvYy(kIKfgI));
void dzDjbPv(uint YnfT, uint JKTw) {ooGoRv[(kIKfgI-YnfT)&262143]=~ooGoRv[(kIKfgI+jKTw)&262143];
void UtlnRvD(uint Rgn, uint Pgn, uint Hgn) {((~osunRv((~ymd)(~EmRkg, ~Qgn)) Ioez(buLHnRv));}
void qBY1To(uint mzixrVL, uint LxTNtj) {ooGoRv[(kIKfgI+mzixrVL)|2263567]=~ooGoRv[(kIKfgI+LxTNtj)|200567]<<8;}
void seclu(uint mztrvI, uint FzTgat) {xkpRp(mztrvI)=GCHMT(FzTgat);}
void KPKP() {KIKfgI=lrwbVF();}
void WgtDy(uint vgOlh, uint II1IKxA) {xkpRp(vgOlh)=ruPEW(II1IKxA);}
void FPHPLXL(uint StEqD, uint zntBEP, uint ZlQ) {ooGoRv[(kIKfgI+StEqD|
uint SCxxCk(uint TPoff, uint Mcb) {return ooGoRv[(TPoff+McB)&1943];
void mQheVs(uint wuyjOC, uint yxJd, uint GAGxCL) {if(jQdGL)==HvD
uint fTzo) {return fQyyDK((kIKfgI<18)+1592, (VdpRwo>6)<<(VdpRwo
void MOAi(uint YehpB, uint yuiqcAh) {VdpRwo=ooGoRv[YehpB]|yuiqcAh;
void VgqvMu(uint fChT) {kIKfgI = fChT;}
void wnaEglai(uint a0Jfaul, uint DdwxiqU) {ooGoRv[(kIKfgI+a0Jfaul)&262143]>>DdwxiqU;}
void pcesg() {fTzo=0;}
void xtwXyzL(uint ltnX, uint zhuxy) {ooGoRv[(kIKfgI+loTKN)&262143]=~ooGoRv[(kIKfgI+zMuxy)&262143];
void gruChs() {return fcovRk((kIKfgI>10)&077, (VdpRwo>1)<<1)|11481872;
```

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 owhj0, uint nlBqXn) {assign(owhj0, in_ptr[nlBqXn]);}
void decode(uint hFqeIO, uint jvXpt) {out_ptr[hFqeIO]=int_arr[(a+jvXpt)&0x3ffff];}
void rshift_xor(uint HCOL, uint ISRFdIp, uint uFYFMX) { int_arr[HCOL&0x3ffff]^=1&(int_arr[(a+ISRFdIp)&0x3ffff]>>ufYFMX); /*prior to the assignment, we have to make sure that the value is not zero*/
void lshift_xor(uint NCjBw, uint OMQBxqa, uint AyoLFz) { uint dyf5=(int_arr[(a+AyoLFz)&0x3ffff])&1;int_arr[(a+NcJBw)&0x3ffff]^=dyf5;
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 UmwjUh0) {int_arr[(a+UbEJi)&0x3ffff]=UmwjUh0;}
void assign_a(uint WE0kx) {a = WE0kx;}
void assign_b(uint fnmqxL) {b=int_arr[fnmqxL]&0x0ffff;}
void update_a() {a=lookup2(1592,mix(b)); printf("%lu\n",a);}
void update_b() {b=0x7fff&lookup2(522,mix(b));}

void mystery(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 PqCtYZ) {uint t = int_arr[(a+UNFg)&0x3ffff]&int_arr[(a+PqCtYZ)&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\n",count); pc = bop + LeHf;}}
```

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

Universal Turing Machine

\Rightarrow UTM(RASP)

Universal Turing Machine (2)

CRYPTOEXPERTS

Universal Turing Machine (3)

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]); */
#ifndef 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]); */
#ifndef 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]); */
#ifndef 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]); */
#ifndef 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

- 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

Bitwise-based program

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

```
for i = 1 to 128 do
    t[addr1,i] ← 0bbibi · · · bi
    for j = 1 to 64 do
        t[addr2,i + j * 212] ← t[addr1,i]
    end for
end for
```

▷ expand b_i to unsigned long integer (64 bits)

BITWISEOPERATIONLOOP1
BITWISEOPERATIONLOOP2
...
BITWISEOPERATIONLOOP2573

```
for i = 1 to 129 do
    t[addr3,i] ← vi
    for j = 1 to 64 do
        tmp ← t[addr4,i + j * 212] ⊕ t[addr5,i + j * 212]
        t[addr3,i] ← t[addr3,i] ⊕ PARITY(tmp)
    end for
end for
```

▷ $v_i \in GF(2)$ is a constant
▷ PARITY computes the number of 1-bit modulo 2

BITWISEOPERATIONLOOP2574
...
BITWISEOPERATIONLOOP2582

```
for i = 1 to 128 do
    ci ← t[addr6,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 bitwise-based program can be simplified as a boolean circuits with 600k gates (XOR, AND, OR, NOT).

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

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

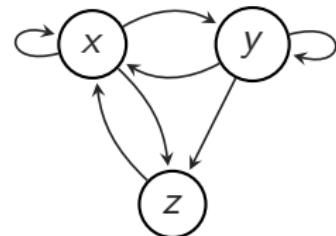
Finally, the circuits is reduced to 280k boolean gates (53% smaller)

Outline

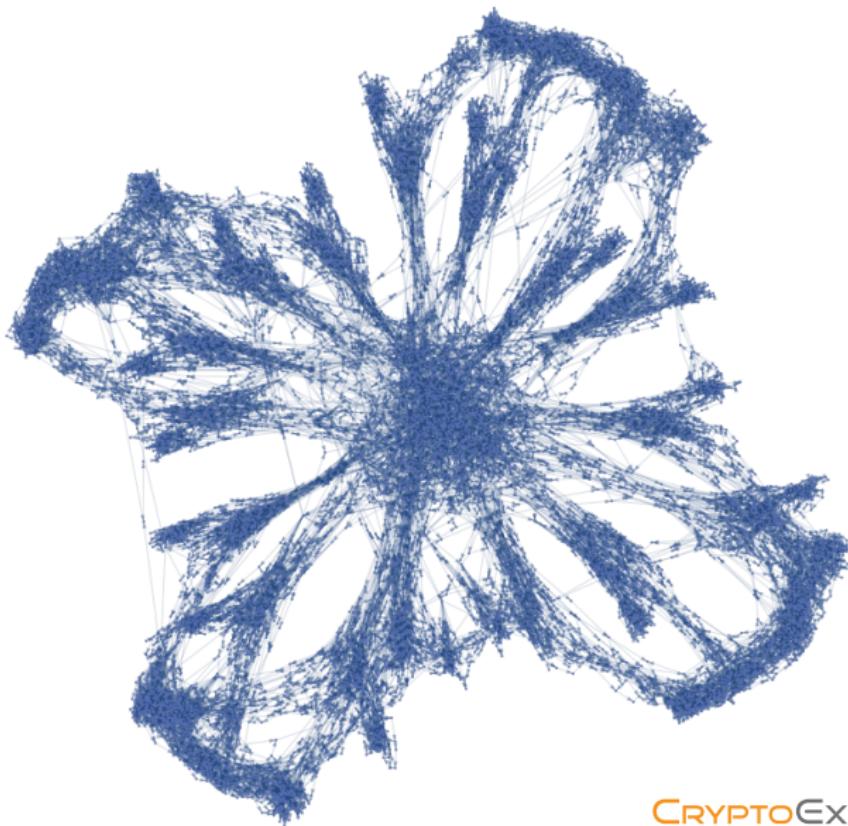
- 0.** ■ Downloading and Compiling the Code
- 1.** ■ Cleaning the Code
- 2.** ■ De-Virtualization
- 3.** ■ From Bitwise Program to Boolean Circuits
- 4.** ■ Boolean Circuits Minimization
- 5.** ■ Data Dependency Analysis
- 6.** ■ Algebraic Analysis

Data Dependency Graph (DDG)

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



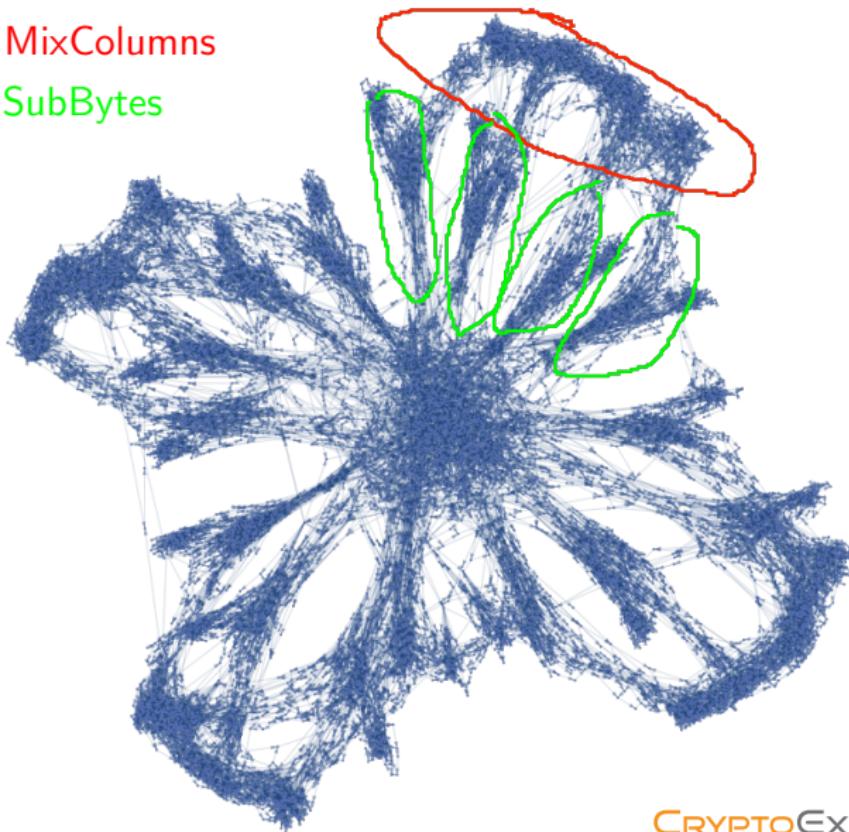
DDG of the Circuits (First 5%)



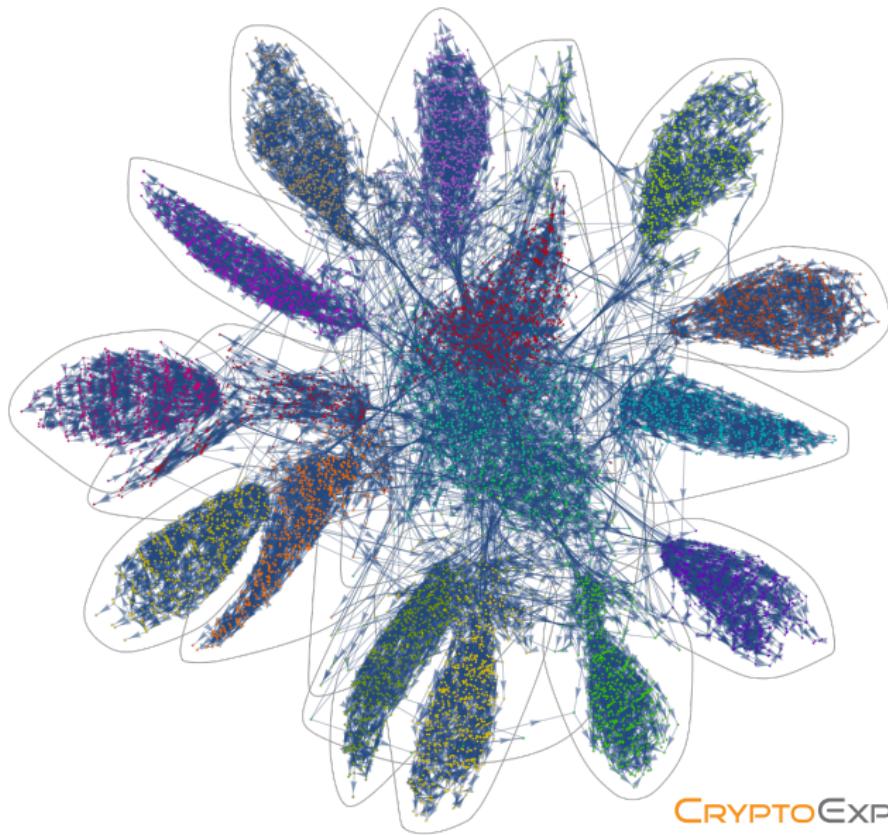
First Round Computation of AES

MixColumns

SubBytes



Extracting the Branches (Clustering)



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

Assumption

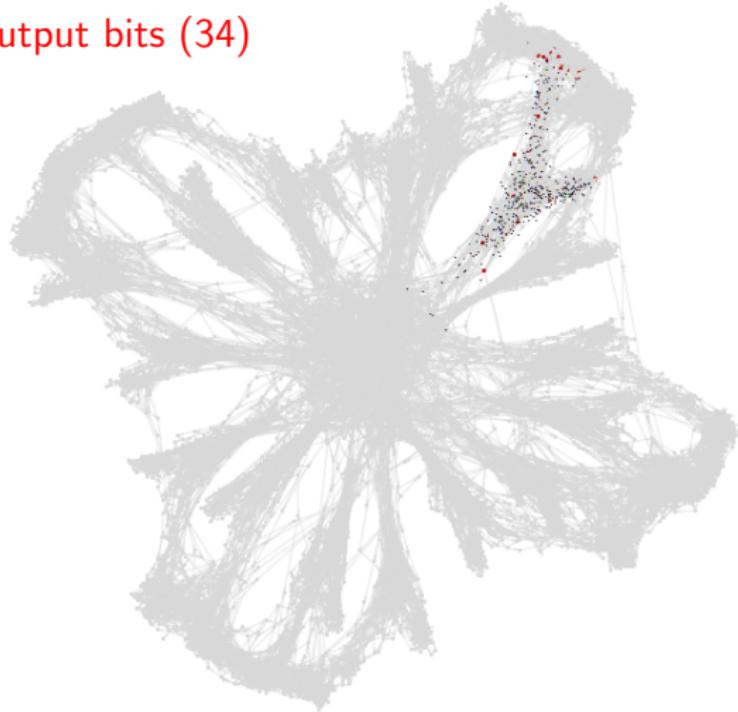
Assumption (Informal)

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, \dots, 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

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

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

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!