# Cold Boot Key Recovery using Polynomial System Solving with Noise

Martin Albrecht & Carlos Cid
Information Security Group, Royal Holloway, University of London

Optimierungsseminar, Zuse Institute Berlin, 10. May 2010

# Outline

# Outline

# Background

Information Security Group

- Cryptography provides the means to accomplish data integrity and confidentiality.
- For hard disk encryption we use **block ciphers** which take a $k$-bit key and encrypt $n$-bit blocks.
- All modern block cipher designs use relatively simple rounds which are repeated $m$ times. In each round $n$ bits of key material are mixed with the current state. Thus, we need to expand the $k$-bit key to $n \times (m+1)$ bits of key material: the **key schedule**.
- We have not seen practical attacks against industry strength block ciphers in decades.
- However, we might be able to exploit side-channel data leakage in order to break data confidentiality.

# Coldboot Attacks I

Information Security Group

- In [7] a method for extracting cryptographic key material from DRAM used in modern computers was proposed.
- Contrary to popular belief information in DRAM is not instantly lost when the power is cut, but decays slowly over time.
- This decay can be further slowed down by cooling the chip.
- Thus, an attacker can
  1. deep-freeze a DRAM module
  2. move it to a target machine which dumps the content to disk
  3. find the most likely key candidate (which is erroneous due to decay)
  4. **use some mechanism to correct those errors**

The technique is called Coldboot attack in literature.

# Coldboot Attacks II

Information Security Group

## Definition (The Coldboot Problem)

We are given

1. $\mathcal{K} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^N$ where $N > n$,
2. two real numbers $0 \leq \delta_0, \delta_1 \leq 1$,
3. $K = \mathcal{KS}(k)$ and $K_i$ the $i$-th bit of $K$.
4. $K' = (K'_0, K'_1, \ldots, K'_{N-1}) \in \mathbb{F}_2^N$ based on the following process:
   - if $K_i = 0$, then let $Pr[K'_i = 1] = \delta_1$ and $Pr[K'_i = 0] = 1 - \delta_1$
   - if $K_i = 1$, then let $Pr[K'_i = 0] = \delta_0$ and $Pr[K'_i = 1] = 1 - \delta_0$.
5. and some control function $\mathcal{E} : \mathbb{F}_2^n \rightarrow \{True, False\}$, which returns true for the pre-image of the noise free version of $K$.

The task is to recover $k$ such that $\mathcal{E}(k)$ returns $True$ or a noise-free $K$.

The Coldboot problem is equivalent to decoding a (non-)linear code with biased noise.

# Coldboot Attacks III

Results in [7]:

| Cipher | $\delta_0$ | $\delta_1$ | Success | Time |
|---|---|---|---|---|
| DES | 0.10 | 0.001 | 100% | – |
| DES | 0.50 | 0.001 | 98% | – |
| AES | 0.15 | 0.001 | 100% | 1s |
| AES | 0.30 | 0.001 | 100% | 30s |

Can we do better and can we recover keys for more complicated key schedules like Serpent?

# Outline

# PoSSo

We define polynomial system solving (**PoSSo**) as the problem of finding a solution to a system of polynomial equations over some field.

**Definition (PoSSo)**

Consider the set $F = \{f_0, \ldots, f_{m-1}\}$ where each $f_i \in \mathbb{F}[x_0, \ldots, x_{n-1}]$.

A solution to $F$ is any point $x \in \mathbb{F}^n$ such that

$$\forall f_i \in F : f_i(x) = 0.$$

Note, that we restrict ourselves to solutions in the base field here.

# Max-PoSSo I

We can define a family of **Max-PoSSo** problems, analogous to the well known Max-SAT family of problems.

`http://en.wikipedia.org/wiki/MAX-SAT`

# Max-PoSSo II

**Definition (Max-PoSSo)**

Find a point $x \in \mathbb{F}^n$ which satisfies the **maximum number** of polynomials in $F = \{f_0, \ldots, f_{m-1}\} \subset \mathbb{F}[x_0, \ldots, x_{n-1}]$.

# Max-PoSSo III

**Definition (Partial Max-PoSSo)**

Find a point $x \in \mathbb{F}^n$ such that for **two sets of polynomials** $\mathcal{H}$ and $\mathcal{S}$ in $\mathbb{F}[x_0, \dots, x_{n-1}]$

- $\forall f \in \mathcal{H} : f(x) = 0$ and
- the number of polynomials $f \in \mathcal{S}$ with $f(x) = 0$ is maximised.

- Max-PoSSo is Partial Max-Posso with $\mathcal{H} = \varnothing$.
- $\mathcal{H}$ for "hard" and $\mathcal{S}$ for "soft".
- Both terms are borrowed from Partial Max-SAT.

# Max-PoSSo IV

Information Security Group

## Definition (Partial Weighted Max-PoSSo)

Find a point $x \in \mathbb{F}^n$ such that

- $\forall f \in \mathcal{H} : f(x) = 0$ and
- $\sum_{f \in \mathcal{S}} \mathcal{C}(f, x)$ is minimized

where $\mathcal{C} : f \in \mathcal{S}, x \in \mathbb{F}^n \to \mathbb{R}_{\geq 0}$ is a **cost function** which

- returns 0 if $f(x) = 0$ and
- some value $> 0$ if $f(x) \neq 0$.

Partial Max-PoSSo is Weighted Partial Max-PoSSo where $\mathcal{C}(f, x)$ returns 1 if $f(x) \neq 0$ for all $f \in \mathcal{S}$.

# Coldboot as Partial Weighted Max-PoSSo

- Let $F_{\mathcal{K}}$ be an equation system corresponding to $\mathcal{K}$.
- Assume that for each noisy output bit $K'_i$ there is some $f_i \in F_{\mathcal{K}}$ of the form $g_i + K'_i$ where $g_i$ is some polynomial.
- Assume that these are the only polynomials involving output bits.
- Denote the set of these polynomials $\mathcal{S}$.
- Denote the set of all remaining polynomials $\in F_{\mathcal{K}}$ as $\mathcal{H}$.
- Define the cost function $\mathcal{C}$ as a function which returns

$$\begin{array}{cr} \frac{1}{\delta_0} & \text{for } K'_i = 0, f_i(x) \neq 0 \\ \frac{1}{\delta_1} & \text{for } K'_i = 1, f_i(x) \neq 0 \\ 0 & \text{otherwise} \end{array} .$$

- Express $\mathcal{E}$ as a polynomial system which is satisfiable for $k$ only and add these polynomials to $\mathcal{H}$.

# Other Applications

**RFID** security is often based on the LPN problem which is easily described as a Max-PoSSo problem.

**Lattices** security often rests on the LWE problem which is easily described as a Max-PoSSo problem.

**Side-Channel** data leakage is often noisy.

**Algebraic Attacks** can be improved by simplifying equation systems using probabilistic equations.

> The family of Max-PoSSo problems has not be studied before as far as we can tell. There is some connection to solving polynomial systems over fixed precision real-numbers.

# Outline

# Mixed Integer Programming I

Integer optimization deals with the problem of minimising (or maximising) a function in several variables subject to linear equality and inequality constraints and integrality restrictions on some or all of the variables.

## Definition (MIP)

A linear mixed-integer programming problem (MIP) is defined as a problem of the form

$$\min_{x}\{c^T x | Ax \leq b, x \in \mathbb{Z}^k \times \mathbb{R}^l\}$$

where

- $A$ is an $m \times n$-matrix ($n = k + l$),
- $b$ is an $m$-vector and $c$ is an $n$-vector.

# Mixed Integer Programming II

Information Security Group

## Example

Maximise $x + 5y$, thus $c = (1, 5)$, subject to the constraints $x + 0.2y \leq 4$ and $1.5x + 3y \leq 4$ where $x \geq 0$ is real valued and $y \geq 0$ is integer valued.

The optimal value for $c^T x$ is $5\frac{2}{3}$ for $x = \frac{2}{3}$ and $y = 1$.

```
sage: p = MixedIntegerLinearProgram()
sage: x, y = p.new_variable(), p.new_variable()
sage: p.set_integer(y[0])
sage: p.add_constraint(x[0] + 0.2*y[0], max=4)
sage: p.add_constraint(1.5*x[0] + 3*y[0], max=4)
sage: p.set_min(x[0],0); p.set_min(y[0],0)
sage: p.set_objective(x[0] + 5*y[0])
sage: p.solve() # work in progress (#8672): allow solver='SCIP'
5.6666666666666661
```

# PoSSo as MIP I

Consider some $f \in \mathbb{F}_2[x_0, \dots, x_{n-1}]$ and let $\mathcal{Z}$ a function that takes a polynomial over $\mathbb{F}_2$ lifts it to the integers. Analogous for elements in $\mathbb{F}_2$.

1. Restrict all $x_i$ to binary values.
2. Evaluate $\mathcal{Z}(f)$ on all $\{\mathcal{Z}(x) \mid x \in \mathbb{F}_2^n, f(x) = 0\}$.
3. Let $\ell$ be the minimum value and $u$ the maximum value.
4. Introduce some integer variable $\frac{\ell}{2} \leq m \leq \frac{u}{2}$.
5. Replace each monomial in $f - 2m$ by a new linearised variable, call the result $g$ and add the linear constraint $g = 0$.
6. For each monomial $t = \prod_{i=1}^{N} x_i$
   - add a constraint $x_i \geq t$ and
   - add a constraint $0 \leq \sum_{i=1}^{N} x_i - t \leq N - 1$.

This is the **Integer Adapted Standard Conversion** [3].

# PoSSo as MIP II

Information Security Group

## Example

Consider $f = ac + a + b + c + 1$

- $\{x \mid x \in \mathbb{F}_2^3, f(x) = 0\} = \{(1,0,0),(0,1,0),(0,0,1),(1,0,1)\}$
- $\ell = 1,\ u = 2$
    1. $g = M + a + b + c + 1 - 2m = 0$
    2. $a \geq M$
    3. $c \geq M$
    4. $0 \leq a + c - M \leq 1$

# PoSSo as MIP III

Information Security Group

```
sage: attach anf2mip.py
sage: B.<a,b,c> = BooleanPolynomialRing()
sage: f = a*c + a + b
sage: bc = BooleanPolynomialMIPConverter()
sage: p = bc.integer_adapted_standard_conversion([f]); p
Mixed Integer Program  ( minimization, ...
sage: p.show()
Minimization:
  x_1 +x_2 +x_3 +x_4
Constraints:
  0 <= -2 x_0 +x_1 +x_2 +x_3 <= 0
  -1 <= x_2 -1 x_3 <= 0
  -1 <= x_2 -1 x_4 <= 0
  0 <= -1 x_2 +x_3 +x_4 <= 1
Variables:
  x_0 is an integer variable (min=0.0, max=1.0)
  x_1 is an boolean variable (min=0.0, max=1.0)
  x_2 is a real variable (min=0.0, max=1.0)
  x_3 is an boolean variable (min=0.0, max=1.0)
  x_4 is an boolean variable (min=0.0, max=1.0)
```

# PoSSo as MIP IV

```
sage: attach anf2mip.py
sage: B.<a,b,c> = BooleanPolynomialRing()
sage: f = a*c + a + b + 1
sage: g = a + c + 1

sage: p = bc.integer_adapted_standard_conversion([f]); p
Mixed Integer Program  (...
sage: p.solve()
1.0

sage: bc.solve([f])
CPU Time: 0.00  Wall time: 0.00, Obj:  1.00
{b: 1, c: 0, a: 0}

sage: bc.solve([f,g],solver='SCIP')
CPU Time: 0.00  Wall time: 0.00, Obj:  1.00
{b: 0, c: 0, a: 1}
```

# Partial Weighted Max-PoSSo as MIP

Information Security Group

We only need to consider Partial Weighted Max-PoSSo because it is the most general case:

- Convert each $f \in \mathcal{H}$ to linear constraints as before.
- For each $f_i \in \mathcal{S}$ add a new binary slack variable $e_i$ to $f_i$ and convert the resulting polynomial as before.
- The objective function we minimise is $\sum c_i e_i$ where $c_i$ is the value of $\mathcal{C}(f, x)$ for some $x$ such that $f(x) \neq 0$.

Any optimal solution $x \in S$ will be an optimal solution to the Partial Weighted Max-PoSSo problem.

# Coldboot as MIP

Coldboot → Partial Weighted Max-PoSSo → MIP

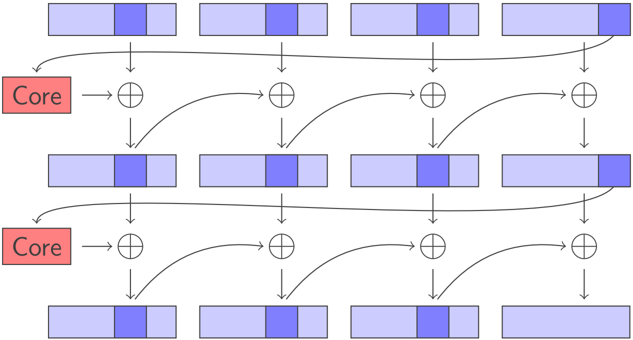
This approach is essentially the non-linear generalisation of decoding random linear codes with linear programming [5].

# Outline

# Simplifications

- We do not model $\mathcal{E}$ since its representation is often too costly; consequently we have no guarantee that the optimal $k$ returned is indeed the $k$ we are looking for.
- We do not include all equations available to us but restrict our attention to a subset (e.g. one or two rounds).
- We may use an "aggressive" modelling strategy where we assume $\delta_1 = 0$ which allows us to promote some polynomials from $\mathcal{S}$ to $\mathcal{H}$. The "normal" modelling assumes $\delta_1 = 0 + \epsilon$.

# AES [4] I



# AES [4] II

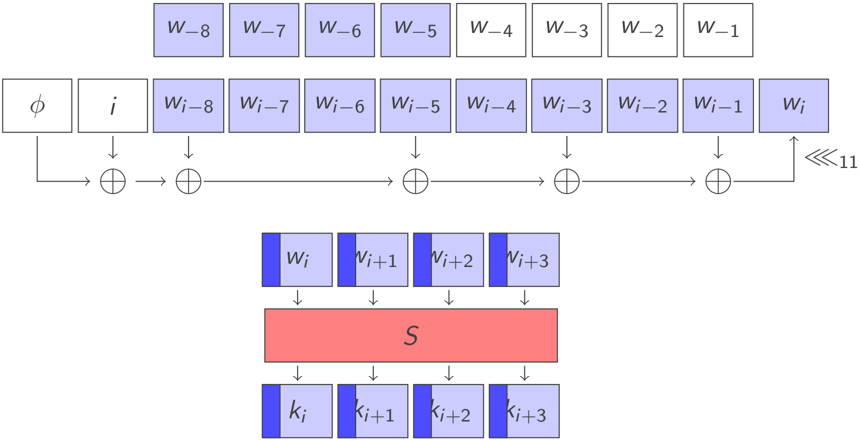
- Most of the key schedule is linear.
- The original key $k$ appears in the output.
- The S-box size is 8-bit (explicit degree: 7).

# AES [4] III

| | | Gurobi [6] | | | | |
|---|---|---|---|---|---|---|
| $N$ | $\delta_0$ | a | #cores | cutoff $t$ | $r$ | max $t$ |
| 3 | 0.15 | − | 24 | $\infty$ | 100% | 17956.4s |
| 3 | 0.15 | − | 2 | 240.0s | 25% | 240.0s |
| 3 | 0.30 | + | 24 | 3600.0s | 25% | 3600.0s |
| 3 | 0.35 | + | 24 | 7200.0s | 10% | 7200.0s |
| 3 | 0.35 | + | 24 | 28800.0s | 30% | 28800.0s |
| | | SCIP (hardlp.set) [1] | | | | |
| 3 | 0.15 | + | 1 | 3600.0s | 65% | 3600.0s |
| 3 | 0.30 | + | 1 | 7200.0s | 45% | 7200.0s |
| 3 | 0.35 | + | 1 | 10800.0s | 10% | 10800.0s |
| 3 | 0.40 | + | 1 | 14400.0s | 0% | 14400.0s |
| 4 | 0.40 | + | 1 | 14400.0s | 10% | 14400.0s |

# Serpent [2] I

Information Security Group



# Serpent [2] II

- All key schedule output bits depend non-linearly on the input.
- The original key $k$ does not appear in the output.
- The S-box size is 4-bit (explicit degree: 3).

# Serpent [2] III

| | | Gurobi [6] | | | | |
|---|---|---|---|---|---|---|
| $N$ | $\delta_0$ | a | #cores | cutoff $t$ | $r$ | Max $t$ |
| 8 | 0.05 | − | 2 | 60.0s | 50% | 16.22s |
| 12 | 0.05 | − | 2 | 60.0s | 85% | 60.00s |
| 8 | 0.15 | − | 24 | 600.0s | 20% | 103.17s |
| 12 | 0.15 | − | 24 | 600.0s | 55% | 600.00s |
| 12 | 0.30 | + | 24 | 7200.0s | 20% | 7200.00s |
| | | SCIP (hardlp.set) [1] | | | | |
| 8 | 0.15 | − | 1 | 3600.0s | 15% | 3600.00s |
| 8 | 0.15 | + | 1 | 3600.0s | 5% | 259.97s |
| 12 | 0.15 | + | 1 | 3600.0s | 40% | 271.47s |
| 16 | 0.15 | + | 1 | 3600.0s | 45% | 1942.27s |
| 12 | 0.30 | + | 1 | 3600.0s | 25% | 3600.00s |

# Serpent [2] IV

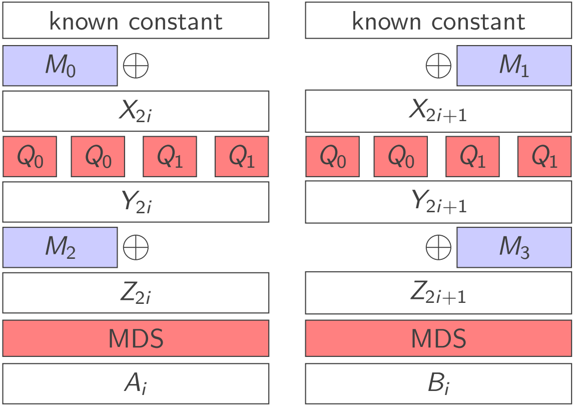
Ad-hoc approach:

- We wish to recover a 128-bit key, so we need to consider at least 128-bit of output.
- On average the noise free output should have 64 bits set to zero.
- In order to consider an error rate up to $\delta_0$, we have to consider

$$\sum_{i=0}^{\lceil \delta_0 \cdot 64 \rceil} \binom{64 + \lceil \delta_0 \cdot 64 \rceil}{i}$$

  candidates and test them.
- If $\delta_0 = 0.15$ we have $\approx 2^{36.87}$.
- If $\delta_0 = 0.30$ we have $\approx 2^{62}$.

# Twofish [8] I




The output of the key schedule is then

$$A_i \boxplus B_i$$

and

$$A_i \boxplus 2 \cdot B_i.$$

# Twofish [8] II

Information Security Group

- The input $k$ $(M_0, \ldots, M_3)$ does not appear in the output.
- All output bits depend non-linearly on the input.
- The S-box $(Q_0, Q_1)$ size is 8-bit (explicit degree: 7)
- There is a modular addition (mod $2^{32}$) at the end.

As of now, we cannot recover the key using mixed integer programming.

# Twofish [8] III

Ad-hoc approach:

- We wish to recover a 128-bit key, so we need to consider at least 128-bit of output.
- On average the noise free output should have 64 bits set to zero.
- In order to consider an error rate up to $\delta_0$, we have to consider

$$\sum_{i=0}^{\lceil \delta_0 \cdot 64 \rceil} \binom{64 + \lceil \delta_0 \cdot 64 \rceil}{i}$$

  candidates and test them.
- If $\delta_0 = 0.15$ we have $\approx 2^{36.87}$.
- If $\delta_0 = 0.30$ we have $\approx 2^{62}$.
- Due to the lack of inner diffusion solving the system for each instance is easy.

# Outline

# Representation

Modular addition modulo $2^{32}$ is used in many cryptographic algorithms to provide non-linearity over $\mathbb{F}_2$. However, over the integers this is linear.

We represent the addition $A \boxplus B = C$ modulo $2^N$ as

$$0 = \sum_{i=0}^{n-1} 2^i A_i + \sum_{i=0}^{n-1} 2^i B_i - \sum_{i=0}^{n-1} 2^i C_i - 2^n$$

for $n \in \{1, \dots, N\}$ and $m \in \{0, 1\}$.

However, this representation may lead to overflows of machine ints and floats.

# Thank you!

# Literature I


Tobias Achterberg.
*Constraint Integer Programming*.
PhD thesis, TU Berlin, 2007.
http://scip.zib.de.

Eli Biham, Ross J. Anderson, and Lars R. Knudsen.
Serpent: A new block cipher proposal.
In S. Vaudenay, editor, *Fast Software Encryption 1998*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer Verlag, 1998.

Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe.
Bivium as a Mixed-Integer Linear programming problem.
In Matthew G. Parker, editor, *Cryptography and Coding – 12th IMA International Conference*, volume 5921 of *Lecture Notes in Computer Science*, pages 133–152, Berlin, Heidelberg, New York, 2009. Springer Verlag.

# Literature II


Joan Daemen and Vincent Rijmen.
AES Proposal: Rijndael, 9 1999.
Available at http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf.

Jon Feldman.
*Decoding Error-Correcting Codes via Linear Programming*.
PhD thesis, Massachusetts Institute of Technology, 2003.

Inc. Gurobi Optimization.
Gurobi 2.0.
http://www.gurobi.com, 2009.

# Literature III

J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten.
Lest we remember: Cold boot attacks on encryption keys.
In *Proceedings of 17th USENIX Security Symposium*, pages 45–60, 2008.

Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson.
Twofish: A 128-bit Block Cipher, 1998.
Available at
http://www.schneier.com/paper-twofish-paper.pdf.