Cold Boot Key Recovery by Solving Polynomial Systems with Noise

Martin R. Albrecht[1] & Carlos Cid[2]

Team Salsa, LIP6, UPMC, Paris, France

Information Security Group, Royal Holloway, University of London, UK

ACNS, 07. June 2011

Outline

Coldboot Attacks

Polynomial System Solving with Noise

Mixed Integer Programming

Application

Outline

Coldboot Attacks

Polynomial System Solving with Noise

Mixed Integer Programming

Application

Coldboot Attacks I

- Recently a method for extracting data from RAM was proposed.
- 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

Definition (The Coldboot Problem)

We are given

$\mathcal{K} : \mathbb{F}_2^n \to \mathbb{F}_2^N$ where $N > n$,

two real numbers $0 \leq \delta_0, \delta_1 \leq 1$,

$K = \mathcal{KS}(k)$ and K_i the i-th bit of K.

$K' = (K'_0, K'_1, \ldots, K'_{N-1}) \in \mathbb{F}_2^N$ with the following distribution:

$$\begin{array}{rclrcl} Pr[K'_i = 0 \mid K_i = 0] & = & 1 - \delta_1, & Pr[K'_i = 1 \mid K_i = 0] & = & \delta_1, \\ Pr[K'_i = 1 \mid K_i = 1] & = & 1 - \delta_0, & Pr[K'_i = 0 \mid K_i = 1] & = & \delta_0. \end{array}$$

and some control function $\mathcal{E} : \mathbb{F}_2^n \to \{True, False\}$, 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 $True$.

Coldboot Attacks III

A bit $K'_i = 0$ of K' is correct with probability

$$Pr[K_i = 0 \mid K'_i = 0] = \frac{Pr[K'_i = 0 | K_i = 0] Pr[K_i = 0]}{Pr[K'_i = 0]} = \frac{(1 - \delta_1)}{(1 - \delta_1 + \delta_0)}.$$

Likewise, a bit $K'_i = 1$ of K' is correct with probability $\frac{(1-\delta_0)}{(1-\delta_0+\delta_1)}$. We denote these values by Δ_0 and Δ_1 respectively.

Coldboot Attacks IV

Cipher	δ_0	δ_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

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.

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

Outline

Coldboot Attacks

Polynomial System Solving with Noise

Mixed Integer Programming

Application

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

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

Max-PoSSo

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

Definition (Partial Weighted Max-PoSSo)

Given two set $\mathcal{H}$ and $\mathcal{S}$ of polynomials. 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 $f(x) \neq 0$.

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

$$\begin{array}{ll} \frac{1}{1-\Delta_0} & \text{for } K_i' = 0, f(x) \neq 0, \\ \frac{1}{1-\Delta_1} & \text{for } K_i' = 1, f(x) \neq 0, \\ 0 & \text{otherwise.} \end{array}$$

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

Outline

Coldboot Attacks

Polynomial System Solving with Noise

Mixed Integer Programming

Application

Mixed Integer Programming I

Integer optimization deals with the problem of minimising a function 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.

PoSSo as MIP

- We can express solving polynomial systems (over $\mathbb{F}_2$) as Mixed Integer Programs.
- We can then use an off-the-shelf MIP solver.
- In this work we use the **Integer Adapted Standard Conversion**.

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.

Partial Weighted Max-PoSSo as MIP

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 usign IASC.
- For each $f_i \in \mathcal{S}$ add a new binary slack variable e_i to f_i and convert the resulting polynomial using IASC.
- 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 $\rightarrow$ Partial Weighted Max-PoSSo $\rightarrow$ MIP

Outline

Coldboot Attacks

Polynomial System Solving with Noise

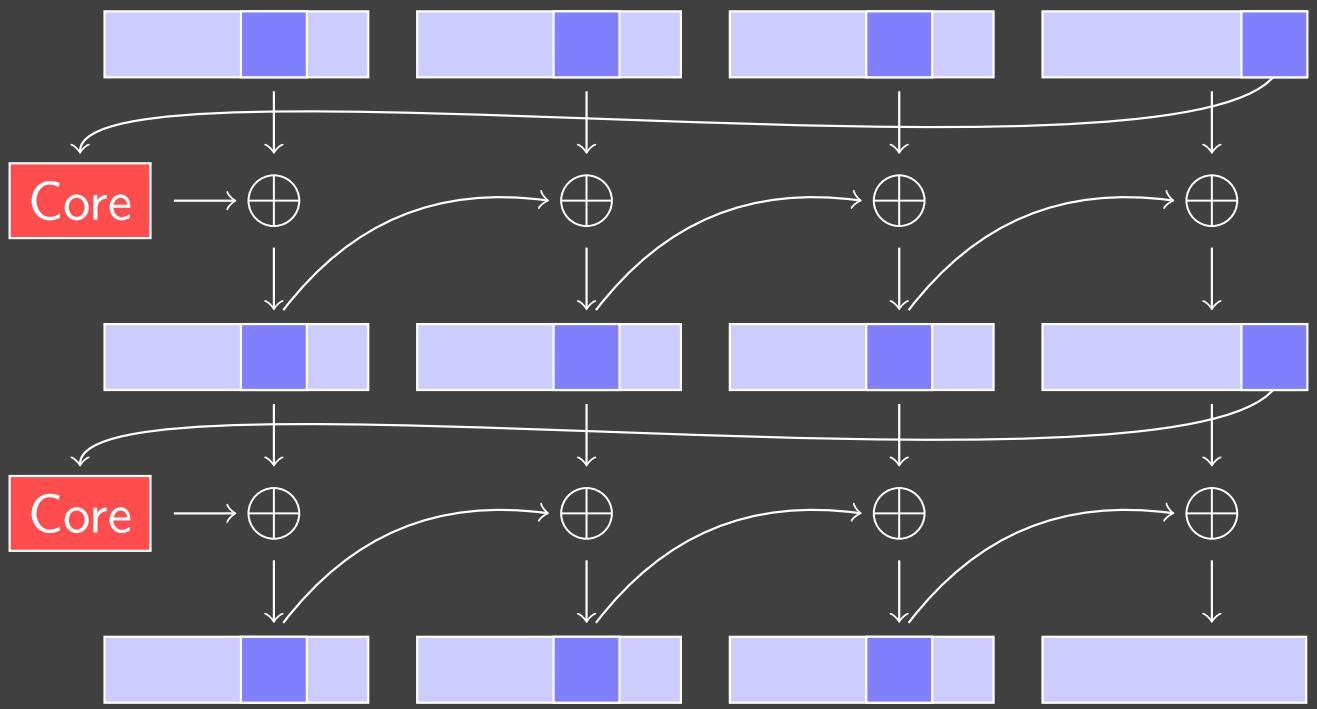
Mixed Integer Programming

Application

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 I



AES II

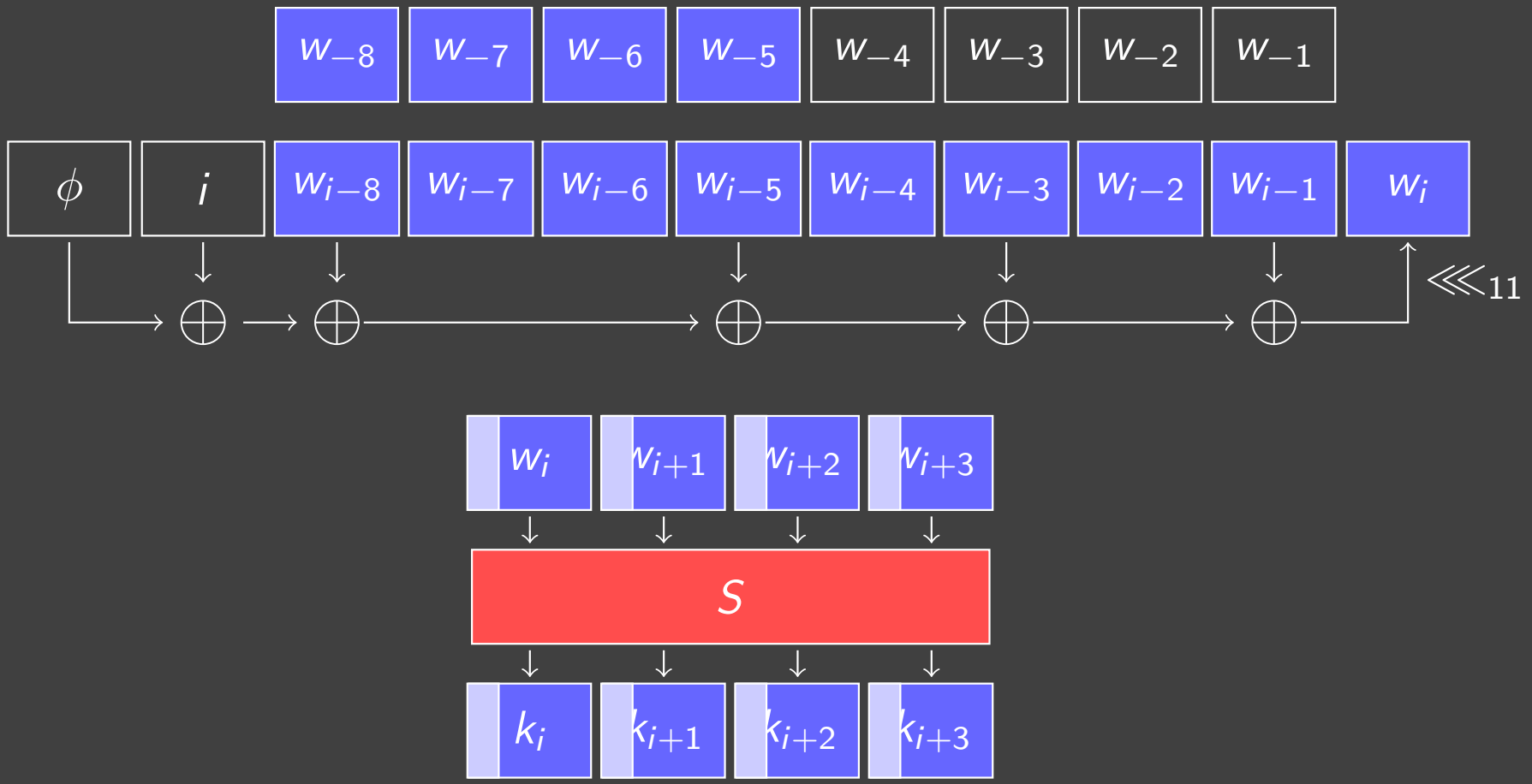
- Most of the key schedule is linear.
- The original key k appears in the output.
- The S-box size is 8-bit (explicit degree: 7).

AES III

N	δ_0	aggr	limit t	r	min t	avg. t	max t
2	0.05	–	3600.00	59%	50.80 s	2124.90 s	3600.00 s
3	0.15	+	60.0s	63%	1.38 s	8.84 s	41.66 s
4	0.15	+	60.0s	70%	1.78 s	11.77 s	59.16 s
4	0.30	+	600.0s	66%	4.81 s	116.07 s	600.00 s
4	0.30	+	3600.0s	69%	4.86 s	117.68 s	719.99 s
4	0.35	+	600.0s	65%	4.66 s	185.14 s	600.00 s
4	0.35	+	3600.0s	68%	4.45 s	207.07 s	1639.55 s
4	0.40	+	600.0s	47%	4.95 s	284.99 s	600.00 s
4	0.40	+	3600.0s	61%	4.97 s	481.99 s	3600.00 s
5	0.40	+	3600.0s	62%	7.72 s	704.33 s	3600.00 s
4	0.50	+	3600.0s	8%	6.57 s	3074.36 s	3600.00 s
4	0.50	+	7200.0s	13%	6.10 s	5882.66 s	7200.00 s

Table: AES considering N rounds of key schedule output.

Serpent I


w_{-8} w_{-7} w_{-6} w_{-5} w_{-4} w_{-3} w_{-2} w_{-1}
ϕ i w_{i-8} w_{i-7} w_{i-6} w_{i-5} w_{i-4} w_{i-3} w_{i-2} w_{i-1} w_i
$\lll 11$
w_i w_{i+1} w_{i+2} w_{i+3}
S
k_i k_{i+1} k_{i+2} k_{i+3}

Serpent 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 III

N	δ_0	aggr	limit t	r	min t	avg. t	max t
12	0.05	–	600.0s	37%	8.22 s	457.57 s	600.00 s
12	0.15	+	60.0s	84%	0.67 s	11.25 s	60.00 s
16	0.15	+	60.0s	79%	0.88 s	13.49 s	60.00 s
$16 \ll 8$	0.15	+	1800.0s	64%	95.52 s	1089.80 s	1800.00 s
16	0.30	+	600.0s	74%	1.13 s	57.05 s	425.48 s
16	0.50	+	1800.0s	21%	135.41 s	1644.53 s	1800.00 s
16	0.50	+	3600.0s	38%	136.54 s	2763.68 s	3600.00 s

Table: Serpent considering $32 \cdot N$ bits of key schedule output

Serpent 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 δ_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 $\approx 2^{62}$.

Thank you for your attention!