

Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints

Gildas Avoine ¹ Pascal Junod ² Philippe Oechslin ³

¹EPFL, Lausanne, Switzerland

²Nagravision SA (Kudelski Group), Cheseaux, Switzerland

³Objectif Sécurité, Gland, Switzerland

Introduction to Time-Memory Trade-Offs

Analysis of Perfect Rainbow Tables

Detecting False Alarms Using Checkpoints

Introduction to Time-Memory Trade-Offs

Goal

Idea

The general idea of time-memory trade-offs is to reduce the time complexity of a given problem by adding memory (or the inverse).

Example: Exhaustive search on a one-way function $S : A \rightarrow B$ in order to find preimages ($N := |X|$).

Extreme cases:

On-line computation: N Storage: 0 Pre-computation: 0

On-line computation: 0 Storage: N Pre-computation: N

Hellman's trade-off (1980): $T = N^2/M^2$ e.g. $T = M = N^{2/3}$.

Pre-Computation Phase

- ▷ Invert $S : A \rightarrow B$.
- ▷ Define $R : B \rightarrow A$ an arbitrary (**reduction**) function.
- ▷ Define $f : A \rightarrow A$ s.t. $f = R \circ S$.
- ▷ Chains are generated from arbitrary values in A .

$$\begin{array}{rcccccccc} S_1 & = & X_{1,1} & \xrightarrow{f} & X_{1,2} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{1,t} & = & E_1 \\ S_2 & = & X_{2,1} & \xrightarrow{f} & X_{2,2} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{2,t} & = & E_2 \\ & & \vdots & & & & & & & & \vdots \\ S_m & = & X_{m,1} & \xrightarrow{f} & X_{m,2} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{m,t} & = & E_m \end{array}$$

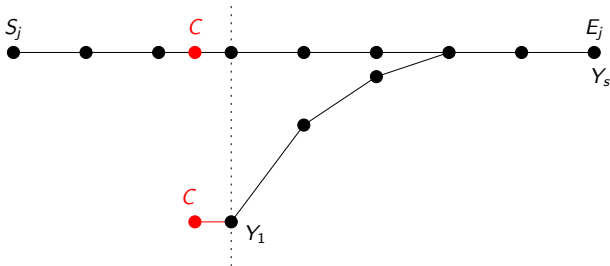
- ▷ Only the **first** and the **last** element of each chain is stored.
- ▷ The generated values should **cover** the set A .
- ▷ Time-memory trade-off techniques are **probabilistic**.

Collisions and Merges

- ▷ Collisions occur during the **pre-computation** phase.
- ▷ We use **several** tables (ℓ) with different reduction functions.
- ▷ We can use one reduction function per column (**rainbow tables**).
 - ▶ If 2 chains collide in different columns, they don't merge.
 - ▶ If 2 chains collide in the same column, the merge can be detected.

On-Line Phase

- ▷ Given one output $C \in B$, we compute $Y_1 := R(C)$ and generate a chain starting at Y_1 : $Y_1 \xrightarrow{f} Y_2 \xrightarrow{f} Y_3 \xrightarrow{f} \dots Y_s$



Sequential Search

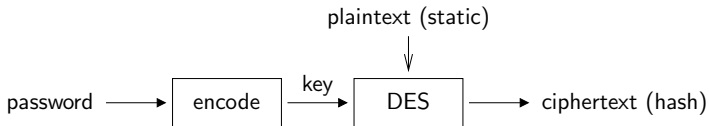
table 1

table 2

...

table ℓ

Cracking Windows (LM Hash) Passwords [Oechslin03]



- ▷ Cracking an alphanumerical password requires
 - ▶ a few **hours** using a classical brute force.
 - ▶ a few **seconds** using a time-memory trade-off.
- ▷ Storing the tables requires about 1 GB.
- ▷ False alarms increases by 125% the expected computation.

See <http://ophcrack.sourceforge.net/>

Analysis of Perfect Rainbow Tables

Probability of Success and False Alarms

- ▶ Probability of success in search k is

$$p_k = \frac{m}{N} \left(1 - \frac{m}{N}\right)^{k-1}.$$

- ▶ The probability of a false alarm at search k (i.e., in column $c = t - \lfloor \frac{k}{\ell} \rfloor$) is

$$q_c = 1 - \frac{m}{N} - \prod_{i=c}^{i=t} \left(1 - \frac{m_i}{N}\right)$$

where $c = t - \lfloor \frac{k}{\ell} \rfloor$, $m_t = m$, and $m_{i-1} = -N \ln(1 - \frac{m_i}{N})$.

Average Cryptanalysis Time

- ▷ The average cryptanalysis time is

$$T = \sum_{\substack{k=1 \\ c=t-\lfloor \frac{k}{\ell} \rfloor}}^{k=\ell t} p_k (W(t-c-1) + Q(c)) \ell + \left(1 - \frac{m}{N}\right)^{\ell t} (W(t) + Q(1)) \ell$$

where

$$W(x) = \sum_{i=1}^{i=x} i \quad \text{and} \quad Q(x) = \sum_{i=x}^{i=t} q_i i.$$

Detecting False Alarms Using Checkpoints

Basic Idea of Checkpoints

Idea

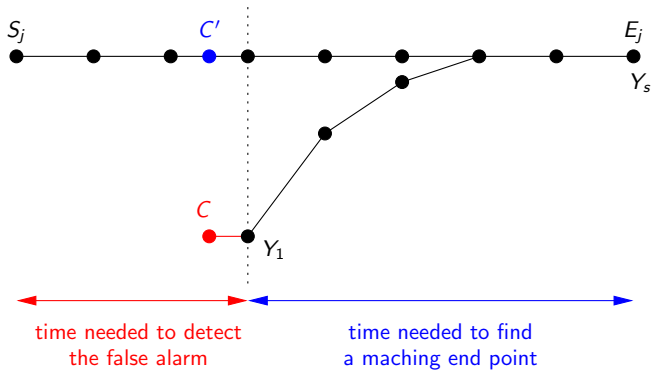
Detect that a value matching a ending point will lead to a false alarm, without regenerating the chain from its starting point.

Definition

Checkpoint

A checkpoint is an index on the chain where a probabilistic test G is applied (e.g. G is a parity test).

Checkpoints



For each chain the result of the test at the checkpoint is stored with the starting and ending points of the chain: $(S_j, E_j, G(\alpha_j))$.

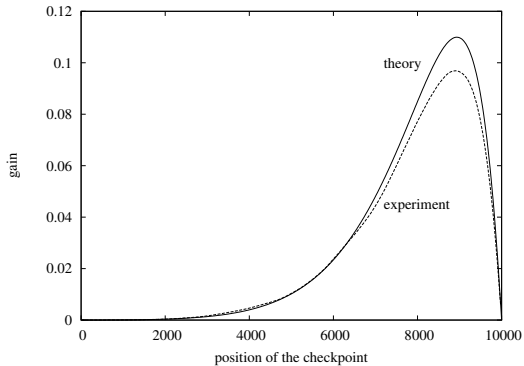
Only one Checkpoint

We define $g_\alpha(s)$ as follows:

$$g_\alpha(s) = \begin{cases} 0 & \text{if there is no checkpoint in column } \alpha \\ 0 & \text{if } (\alpha + s) \leq t, \text{ i.e. the chain does not reach col. } \alpha \\ \Pr\{G(X_{j,\alpha}) \neq G(Y_{\alpha+s-t}) \mid X_{j,\alpha} \neq Y_{\alpha+s-t}\} & \text{otherwise} \end{cases}$$

We can now rewrite $Q(x) = \sum_{i=x}^{i=t} i(q_i - q_\alpha \cdot g_\alpha(t - i))$.

Gain With Only One Checkpoint



Generalization to Several Checkpoints

$$Q(x) = \sum_{i=x}^{i=t} i \left(q_i - q_i \cdot g_i(t-i) - \sum_{j=i+1}^{j=t} \left(q_j \cdot g_j(t-j) \prod_{k=i}^{k=j-1} (1 - g_k(t-k)) \right) \right)$$

A Fair Comparison

- ▶ Storing checkpoints **consumes memory**.
- ▶ A trade-off between memory used to **store chains** and memory used to **store checkpoints** must be found.
- ▶ We now define **memory cost** and **time gain**. Let M , T , N and M' , T' , N' be the parameters of two trade-offs respectively. We define σ_M and σ_T as follows:

$$M' = \sigma_M \cdot M \quad \text{and} \quad T' = \sigma_T \cdot T.$$

Gain With Several Checkpoints

Number of checkpoints	1	2	3	4	5	6
Cost (memory)	0.89%	1.78%	2.67%	3.57%	4.46%	5.35%
Gain (time) storing chains	1.76%	3.47%	5.14%	6.77%	8.36%	9.91%
Gain (time) storing checkpoints	10.99%	18.03%	23.01%	26.76%	29.70%	32.04%
Optimal checkpoints	8935	8565 9220	8265 8915 9370	8015 8655 9115 9470	7800 8450 8900 9250 9550	7600 8200 8700 9000 9300 9600
	± 5	± 5	± 5	± 5	± 50	± 100

Cracking Windows passwords with parameters $N = 8.06 \times 10^{10}$, $t = 10'000$, $m = 15'408'697$, and $\ell = 4$.

Conclusion

Conclusion

- ▶ Checkpoints are **efficient** and easy to implement.
- ▶ Checkpoints can be sometimes used **for free**.
- ▶ **Ophcrack v.3** will implement checkpoints.
- ▶ Checkpoints open a **new way** to explore.