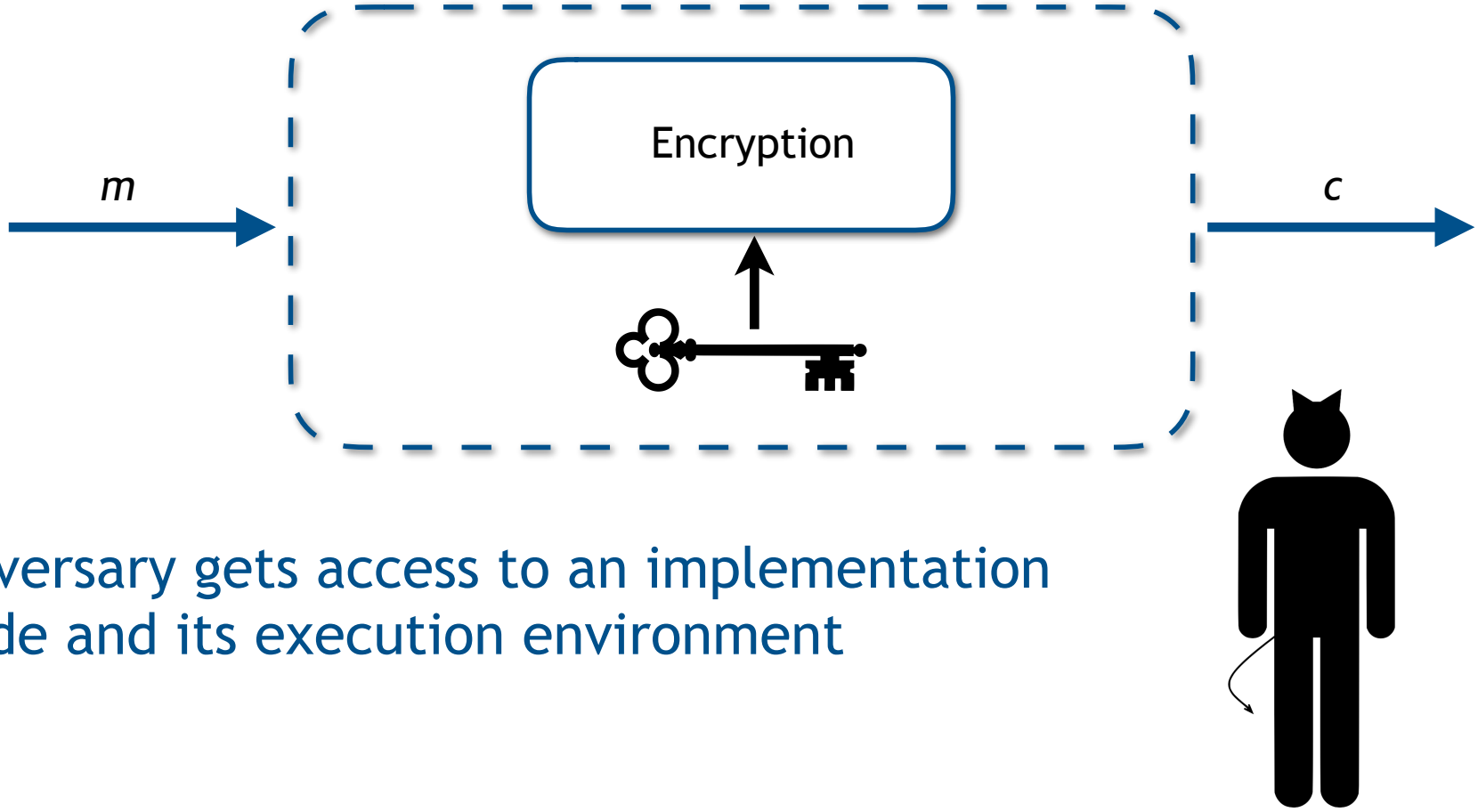
Security Assessment of White-Box Design Submissions of the CHES 2017 CTF Challenge

Estuardo Alpírez Bock and Alexander Treff

White-box attack scenario



Adversary gets access to an implementation code and its execution environment

→ WB Cryptography aims to provide security even under such attack threats

Key extraction attacks

Given the strong adversarial capabilities, white-box programs need to implement countermeasures against key extraction attacks

Some design strategies for white-box implementations of AES have been proposed but also broken

In recent years, powerful attacks such as differential computational and differential fault analysis have been performed on white-box implementations.

Designing a white-box AES implementation which remains secure against key extraction attacks is clearly a very difficult task

CHES CTF Challenge

In this paper

We assess the security of all design candidates of the WhibOx Contest by performing a line of attacks on them

We aim to understand how the candidates can be broken and specially their robustness against automated attacks

Experiments performed by Alexander Treff while doing an internship at Riscure

Our assessment methodology can lead to a more unified way of analysing the security levels provided by a white-box design

The competition

Competition rules

Designers are invited to submit white-box implementations of AES-128

Implementation language must be C, without includes, libraries, etc

Size and runtime restrictions:

Source code	≤	50MB
Binary	≤	20MB
Runtime	≤	1s

Attackers are invited to break the implementations.

The longer an implementation remains unbroken, the more points it gets

Competition results

Submissions:

94 design candidates were submitted
13 remained unbroken for at least 24 hours and earned > 0 points

All broken

Winning challenge: adoring_poitras by Alex Biryukov and Aleksei Udovenko from the University of Luxembourg

Remained unbroken for 28 days

Broken by the CryptoExperts team [1]

[1] Goubin, Paillier, Rivain, Wang: How to reveal the secrets of an obscure white-box implementation, J. of Cryptographic Engineering

Our assessment

Our assessment

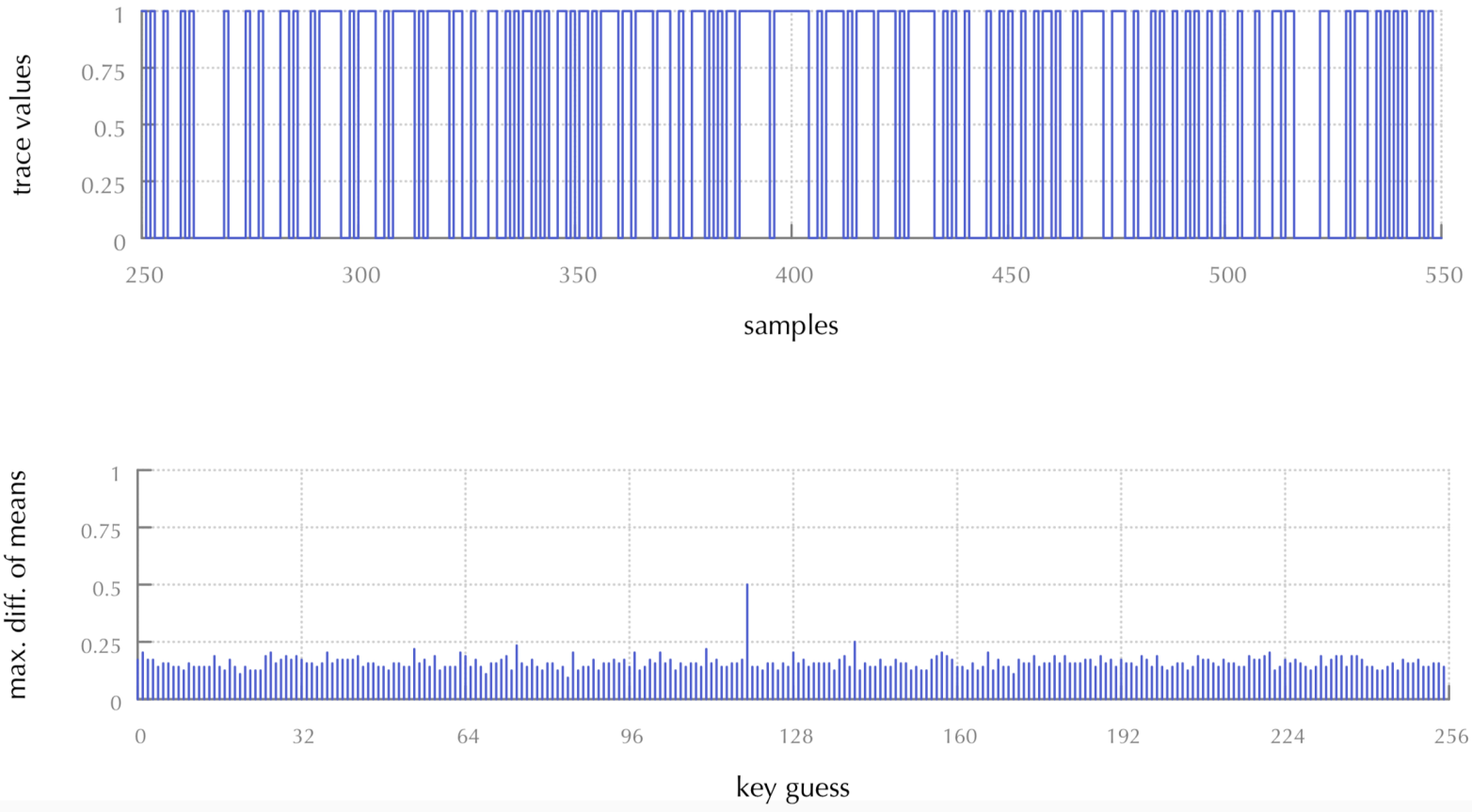
How many challenges can be broken via automated attacks without reverse engineering efforts? Which attacks are effective on which challenges?

Attack classification:

- Automated (DCA, DFA, Higher-order DCA)
- Automated after small modifications
- Automated Robust (remain unbroken in our assessment)

Differential computation analysis

- Software counterpart of differential power analysis



Differential computation analysis

A total of 50 design candidates were vulnerable to a fully automated DCA attack

37 designs were reference AES implementations (w/o white-box countermeasures)

13 designs implemented code obfuscation techniques or were table based designs (following the approach by [2])

All designs were broken within minutes during the competition

[2] Chow, Eisen, Johnson, van Oorschot: White-box cryptography and an AES implementation, SAC 2002

DCA after modifications

Some designs implemented countermeasures against DCA such as:

- Dummy operations —> misalignment and artificially enlargement of the traces
- Inconsistent implementation of round functions

```
switch (*((int *)_obf_3_MOD_AES_encrypt_$pc[0])) {
case 47:
    // cT() is computationally expensive
    // but always computes the same value
    *((unsigned long *)(_obf_3_MOD_AES_encrypt_$locals + 856)) = cT();
    break;
// several more cases, all similar to the one above
}

// we replace the computation with its result
u32 cT() { return 1262335309; }
```

pensive_shaw

5 more candidates can be broken after simple modifications

Differential fault analysis

- In case DCA did not succeed, we apply DFA
- Some designs resisted DCA by artificially blowing up the number of samples recorded per trace
  - In DFA, we induce faults by flipping bits towards the end of the computation, and analyse how the faults are reflected on the outputs
  - We use the DFA script from the Side-Channel Marvels repository [3]
  - Could break 14 designs in a fully automated way
  - For some designs, we needed about an hour to attack them

[3] https://github.com/SideChannelMarvels

Manual DFA

Some designs implemented countermeasures against DFA, e.g. redundant computations

But they could be removed manually

For other designs, we needed to add lines of code for identifying the correct spot for fault injection

```
void AES_128_encrypt(char* ciphertext, char* plaintext)
{
  int COUNTER = atoi(ARGV[1]); // injected code
  for (int i = 0; i < 60000; i++) {
    if (COUNTER == i) { // injected code
      continue;
    }
    func(a,b,c,d); // original WB code
  }
}
```

7 more candidates could be broken via manual DFA

Second-order DCA

- The challenge priceless_stallman was resistant to DFA and DCA
  - Resisted DCA via masking based on the input message
- We could attack this challenge via second-order DCA, performed in a similar style as second-order DPA [4]
- Running this analysis took about 16 hours
  - The attack remained unbroken for only 1:18 hrs during the competition time

[4] Bogdanov, Rivain, Vejre, Wang: Higher-order DCA against standard side-channel countermeasures, Cosade 2019

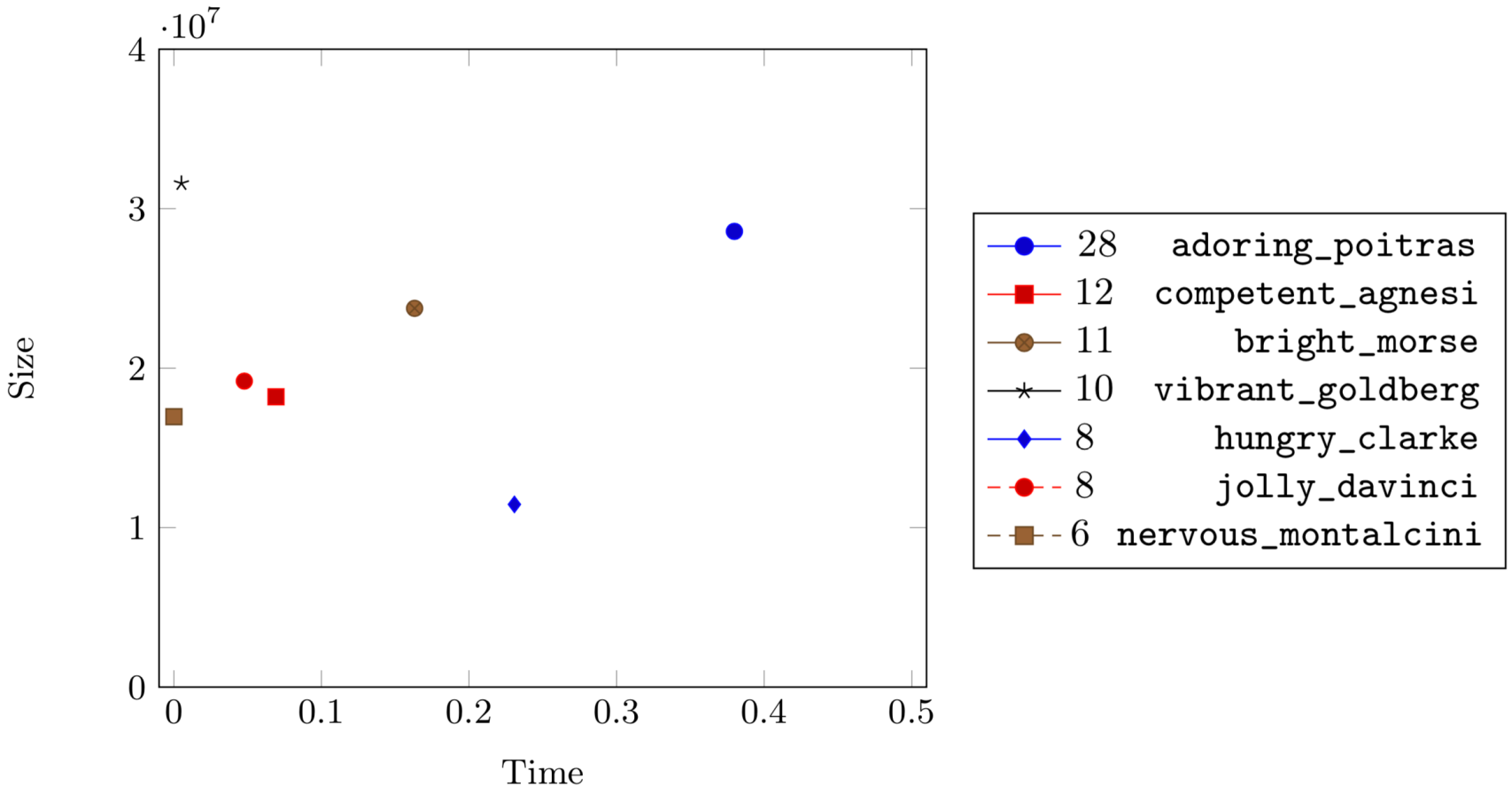
Unbroken challenges

rank	name	id	size	runtime	time unbroken
1	adoring_poitras	777	27.252	379.83	685:42
2	competent_agnesi	815	17.359	6.923	290:15
3	bright_morse	753	22.649	163.14	283:50
4	vibrant_goldberg	877	30.126	5.15	254:59
5	hungry_clarke	845	10.925	230.76	196:44
6	jolly_davinci	751	18.299	47.77	190:09
7	nervous_montalcini	644	16.17	0.07	139:19
8	sad_goldstine	786	10.401	143.83	61:09
9	mystifying_galileo	84	19.236	114.59	32:33
10	elastic_bell	49	20.709	261.05	27:11
11	practical_franklin	49	15.527	2.58	24:01
12	agitated_ritchie	44	22.946	20.33	24:00
13	clever_hoover	32	18.319	0.97	20:14
14	gallant_ramanujan	153	0.898	0.04	15:15
15	peaceful_williams	47	11.950	2.29	11:47
16	eager_golick	572	38.146	83.53	06:22

Earned points during competition

No points earned

Top 8 challenges



Top 8 challenges

For practical use-cases we'll try to achieve fast, small sized and secure white-box programs

In some cases we might be satisfied if a white-box program can remain unbroken for several days

—> update the white-box before an attacker breaks it

—> update it such that the new version follows a different design strategy as the old one

In this competition, the second ranked challenge competent_agnesi, by Leandro Marin from the University of Murcia and Phillips provided the most interesting numbers within this context

Going forward

2019 edition

—> winning challenge remained unbroken for 51 days

—> 2 other challenges remained unbroken for 50 and 30 days

New attacks

Extensions of automated attacks have been presented, e.g. in

[5] Rivain and Wang: Analysis and improvement of differential computation attacks against internally -encoded white-box implementations, CHES 2019

[6] Goubin, Rivain and Wang: Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks, CHES 2020

[7] Alpirez Bock, Bos, Brzuska, Hubain, Michiels, Mune, Sanfelix Gonzalez, Teuwen and Treff: White-box cryptography: don't forget about grey-box attacks, J. of Cryptology 2019

New ideas for countermeasures have also followed, e.g.

[8] Sekar, Eisenbarth,Liskiewicz: A white-box masking scheme resisting computational and algebraic attacks, eprint 2020/443

Improvements for our assessment

Our assessment could provide a more broad overview of the robustness of a design if we

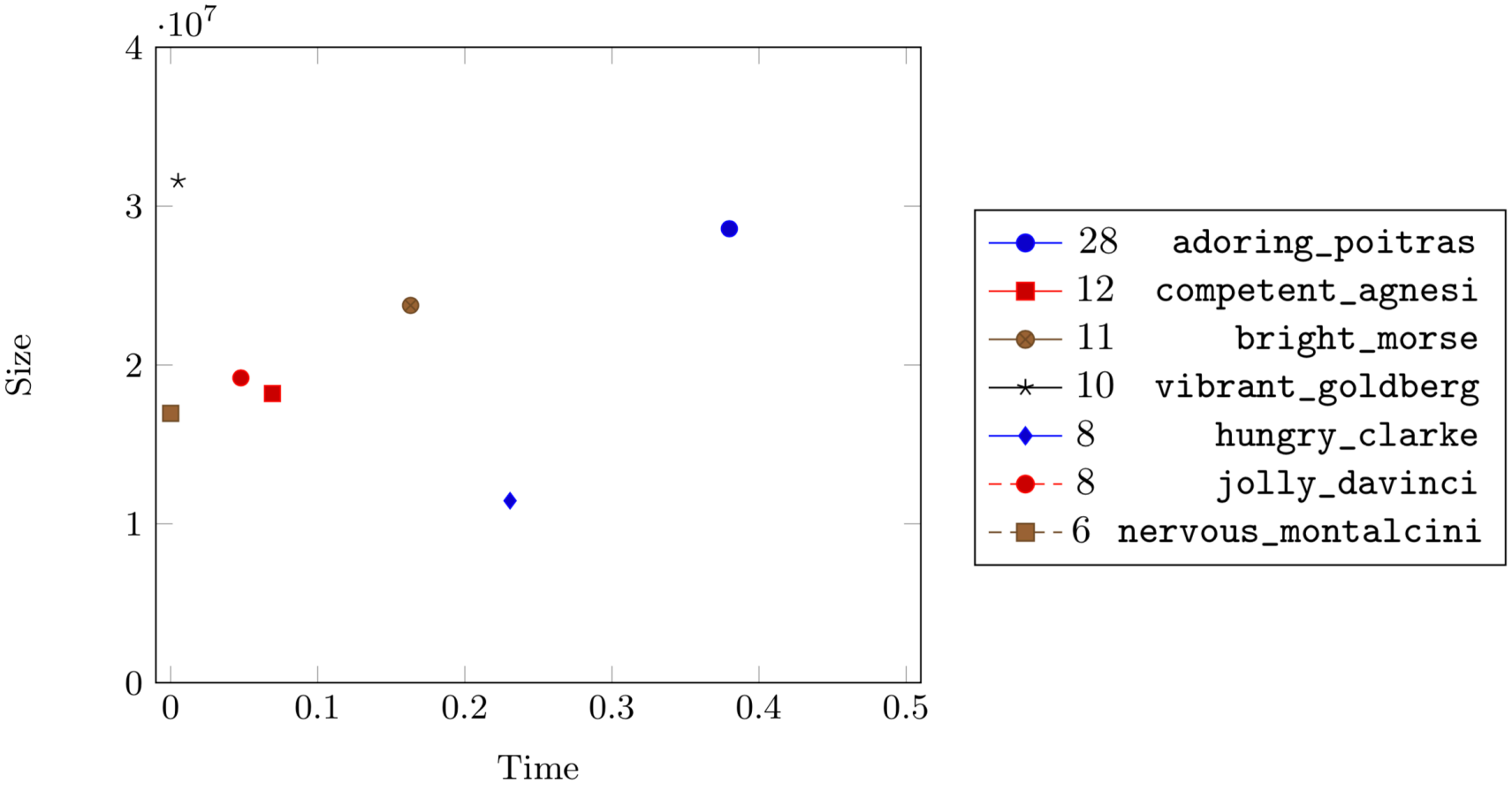
—> Integrate the new attacks as part of our assessment

—> Test all attacks on all candidates

—> Try to standardise what it means to need *only small reverse engineering efforts*

—> Standardise a grading system for the designs: provide points according to the attacks they are resistant to, but also according to their performance

Such assessments could be useful for people in the industry and academia



Thank you for your attention!