AMD tag overhead in a MPC outsourcing environment

Christian Bobach, 20104256

December 20, 2016

Abstract

TODO: WRITE abstract

Contents

1 Intorduction **1**

2 Experiment **2**

2.1 Circuits . . . 3

2.1.1 TAG-circuit . . . 3

2.1.2 ID-circuit . . . 4

2.1.3 Soldering . . . 4

2.2 Results . . . 5

2.3 Discussion . . . 7

3 Conclution **7**

4 Continiued work **8**

A Data **9**

1 Intorduction

In this paper I have studied the paper [JNO16] and made experiments to mesure the overhed added to the computation by securing the input with an algebraic manipulation detection(AMD) code.

TODO: Introduction to paper [JNO16]

In the paper they propose a framework for outsourcing secure multiparti computation(MPC) between any number of clients and workers with minimal overhead. The framework ensures input privacy is preserved and that the output is correct. This is done independant of the underlying MPC protocol. Their solusion minimize the work neede done by the client, while they try to minimize the overhead on the workers. For correctnes of output from the framework it is enough for one client to only trust one worker. This hes the effect hat the framework is not ensured to terminate as one worker can deadlock the system.

They propose a way to detect a disonest worker and not to use that worker next time running the computation.

TODO: Introduction to problem and solution

Because the workers has the possibility to add an error ϵ to the input of a client or before returning the result, there is a need for a tag to ensure the detection of such an addition. In the paper they propose usage of an AMD code to ensute that a worker cannot create an virable tag $\tilde{t} = Tag(\tilde{k}, \tilde{x})$ where $\tilde{t} = t + \epsilon_t$, $\tilde{k} = k + \epsilon_k$ and $\tilde{x} = x + \epsilon_x$. If this is possible a corrupted worker could chose $\tilde{x}$ and creade a matching $\tilde{t}$ such that the protocol will not abort. This problem is exactly what the AMD code can overcome.

Then before the worekers output the secret shere of the function computed. They check that the entegrity of the tag is correct. A corrupted worker cannot change its share of the tag without breaking the security, and therby terminating the execution. This will reveal the corrupted worker and that will not be used in later computations.

TODO: Introduction to experiment

It is the overhead of the added tag that I have studied in this paper. In the paper they state that overhead is not big, but how big should the function computed be before the addition of the tag is negligabel. That is what I try to anwser in this paper.

2 Experiment

The project was done using the duplo framework which is a framework for doing multiparti computations in a two parti setting. The framework is used to construct the two parties communicating in the experiments. The framework allows us to time the comunication. The two parties runs through six different phases during the execution. The six phases are setup, preprocess, solder, prepare, evaluate and decode. These phases give the possibility to time the computation in the different phases. After the six phases the parties hold their output computed based on their input and the circuit of the computation.

TODO: How was the experiemnts done

The experiments in this paper is done using a composed circuit run through the framework. The experiemnt used two different composed circuit as inputs both of them consisting of multiple subcircuits. The first composed circuit we will call C_{ITA} consists of an ID-, a TAG- and a variable number of AES-circuits. The other composed circuit we will call C_{IA} consists of an ID-circuit and a variable number of AES-circuits. The TAG-circuit will be denoted C_T. In the framework used it is not possible to time the different subcircuits of the composed circuit. Therefore both C_{ITA} and C_{IA} is run multiple times to get an avarage of the execution time with and without C_T. From this we can get an estimate of how much time is used on the C_T by calculating $C_{ITA} - C_{IA}$. When we have the time for the C_T it is easy to get the fraction of the time is used on C_T compared to the composed circuit by calculating $\frac{C_T}{C_{ITA}}$. This allow us to figure out how much time C_T adds of overhead. Thereby we get an idea of how big a circuit should be before we can negiligate the overhead added by the AMD tag proposed in [JNO16].

Before the expreiments could begin we needed the circuits that should be used. In the following sections I will introduce these circuits and why they were needed.

2.1 Circuits

The duplo framework had an AES-circuit that could be used as "the big computation", but we needed an ID- and a TAG-circuit before we could do the experiments. By "the big computation" is ment, a computation that could give an idea of when C_T could be negligable. Here we use multiple AES circuits as the variable to see when this will happen. The ID-, TAG- and AES-circuit's would be soldere togeter to form the composed circuit that could be timed in the framework.

The input circuits are encoded using a slide modification of Nigel Smart's[1] circuit encoding. To generate the circuits I used a modifyed version of the Fariplay circuit generator tool[2].

2.1.1 TAG-circuit

The TAG circuit is made such that it forfills the requirements for the tag proposed in [JNO16].

TODO: Security of the tag: do the math

Such that the correctness of the output is guaranteed.

TODO: Description of tag circuit

C_T is constructed such that given 384 bit inputs from bothe patries, x_1, k_1, s_1 and x_2, k_2, s_2 where x_i is the input to the calculation, k_i is the key for the tag and s_i is the seed where $|s_i|$ is the security parameter, for $i = 1, 2$. The tag is calculated as follows

$$\begin{aligned}
Tag(x_1, k_1, s_1, x_2, k_2, s_2) &= \begin{bmatrix} k_{1,1} & k_{2,1} & \dots & k_{1,128} & k_{2,128} \\ k_{1,2} & k_{2,2} & \dots & s_{1,1} & s_{2,1} \\ \vdots & & & & \\ s_{1,1} & s_{1,2} & \dots & s_{1,128} & s_{2,128} \end{bmatrix} \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{1,128} \\ x_{2,128} \end{bmatrix} \\
&= \begin{bmatrix} x_{1,1}k_{1,1} + x_{2,1}k_{2,1} + \dots + x_{1,128}k_{1,128} + x_{2,128}k_{2,128} \\ x_{1,1}k_{1,2} + x_{2,1}k_{2,2} + \dots + x_{1,128}s_{1,1} + x_{2,128}s_{2,1} \\ \vdots \\ x_{1,1}s_{1,1} + x_{2,1}s_{2,1} + \dots + x_{1,128}s_{1,128} + x_{2,128}s_{2,128} \end{bmatrix} \\
&= \begin{bmatrix} t_0 \\ \vdots \\ t_{128} \end{bmatrix}
\end{aligned}$$

This calculation was converted into a circuit such that if $k'_i = k_i || s_i$ and$t'_{i,j,l} = x_{i,l} \; AND \; k'_{i,l+j}$ then $t_j = t'_{1,j,1} \; XOR \; t'_{2,j,1} \; XOR \; \dots \; XOR \; t'_{1,j,128} \; XOR \; t'_{2,j,128}$,

[1]Nighel's encoding can be found here: `https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/`

[2]The circuit generator can be found here: `https://github.com/cbobach/circuit-generator`

for $i = 1, 2$, $j = 0, \ldots, 128$ and $l = 1, \ldots, 128$. Since AND has the structore of multiplication in binary and XOR the structure of addition, as can be seen in table 1.

l-wire	r-wire	l AND r
0	0	0
0	1	0
1	0	0
1	1	1

(a) AND -gate

l-wire	r-wire	l XOR r
0	0	0
0	1	1
1	0	1
1	1	0

(b) XOR -gate

Table 1: Logical gates

Constructiong the circuit like this result in a circuit with 768 input and 129 output wires. Internal C_T consist of 66687 wires and 65919 gates where 33024 of these gates are the expensive AND gates. No optimization has been done to reduce the numer of AND gates in the circuit.

2.1.2 ID-circuit

The ID-circuit was needed since the framework did not support that the inputwires of the composed circuit could go to multiple subcircuits. Therefor we needed a input circuit that could overcome this. Because of the construction of the framework a subcircuit needed to consist of bothe gates and wires.

The ID-circuit, denoted C_I is a circuit where the input and output are the same. Because a subcircuit needed to have gates it was first constructed by using two layers of $NAND$-gates. This was later optimized to use one layer of AND gates instead. This was done by takes each input wire and AND it with itself to get the same output as input which is ilustraded by table 1.

Since the input of the composed circuit is based on the input of C_A, the key size and the security parameter of C_T. The number of input wires to C_I is the same as C_{ITA}. As we use 128 bit AES and 128 bit as key and security parameter C_I has 768 inputwires in total, 384 from each parti. This gives us an additionl 768 AND-gates and 1536 wires that C_I adds to the composed circuit.

2.1.3 Soldering

TODO: Description of the soldering phase: parallel, input to output

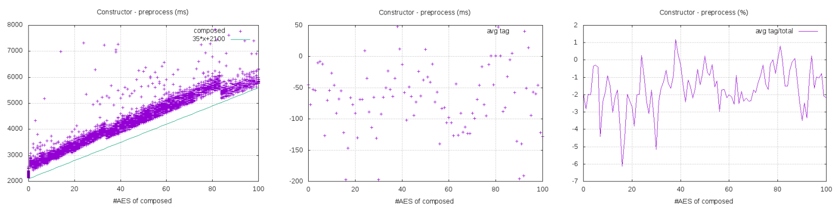
In each run of the two parti computation the different circuits is soldered together to cunstruct the composed circuit. Since aes is used as the computation of the experiment then input frm the two parties shouls be maped correctly at the same time we had the tag circuit that uses the complete input. The composed circuit sonsist of two layers of circuits. In the first layer we have C_I such that we have the same output of layer one as input. In the second layer we have C_T as the first circuit. This circuit uses the complete output from C_I as input and output the tag vektor. The rest of the circuits of the second layer consist of C_A's ranging from 0 and upwards, such that they are soldered to C_I in parallel. Since the C_A's does not use the complete output from C_I the correct output wires need to be soldered to the correct inputwires of the C_A's. As the input to C_A is the first 128 bit outputwires from both parties in C_I thise should be soldered to the such that the outputwires 0 to 127 and 384 to 511 are used as input to each of the C_A's. By soldering multiple C_A's in to the composed circuit we obtain a bigger circuit. That we can use to mesure the tag overhead.

For the framework to work it is important to keep track of which wire indexes goes to which circuit an wich indexes should be on the new outputwires. This can be found in the source code for the duplo framework found in the repository for this project.[3] The code relevant for the soldering of circuits is found in the header file.

2.2 Results

In this section I will try to explain what we can see from the timing of the different phases when running the composed circuit in the framework. All the results can be found in appendix A. In the following section will I only include the plots that are interresting.

First of all we can eliminte the setup phase, since that does not dependant on the circuit but is a technical phase were the parties are prepared. So the first interristing phase to look at is the preprocess phase.



(a) (b) (c)

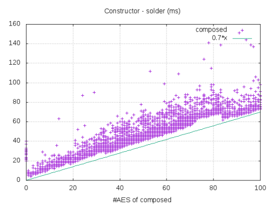
Figure 1: The preprocess phase

In figure 1a we see that the preprocessing time of the composed circuit looks linear to de number of AES subcircuits. We also see that the total time used preprocessing starts around 2-3 seconds without any AES subcircuits and goes up to between 6 to 7 seconds with 100 AES subcircuits. Looking at figure 1b we see that for the most plot is less then 0. This implies that the tag part of the circuit is such small that it is not mesurable. At figure 1c we also see that the fraction of time used preprocessing the tag part of the circuit is minimal and below 1%.

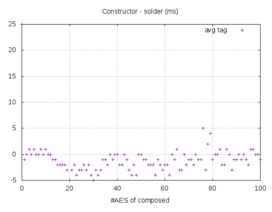
In the solder phase of the execution we see a difference between the two parties. In the figure we see that both 2a and 2d are lineare encreasing. It looks like there is a factor 10 in difference in the slope between them. Then looking at the average time used on the tag in the plots 2b and 2e we see that this looks like a constant less than 1. Which gives us a fraction as seen in 2c and 2f is decreasing on the spikes and over the number of AES circuits in the composed is going towards something negligable.

In the prepare phase the two parties plots are close to equivalente. This time we see that the time used on the composed circuit in figure 3a is decreasing and

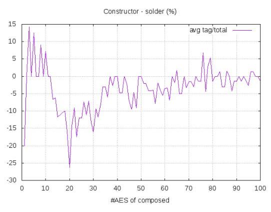
[3]Code for this experiment can be found on github here: `https://github.com/cbobach/mpc-outsourcing`



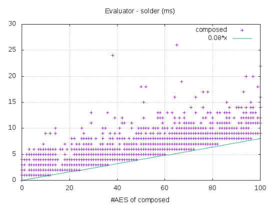
(a)



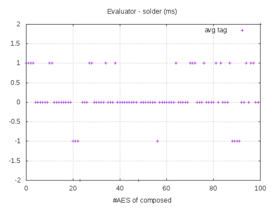
(b)



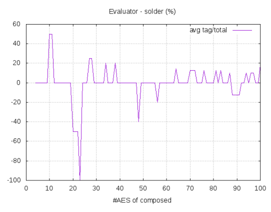
(c)



(d)

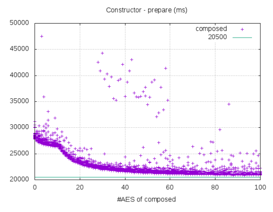


(e)

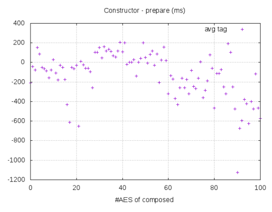


(f)

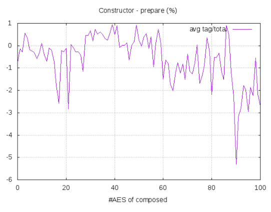
Figure 2: The soldering phase



(a)



(b)



(c)

Figure 3: The prepare phase

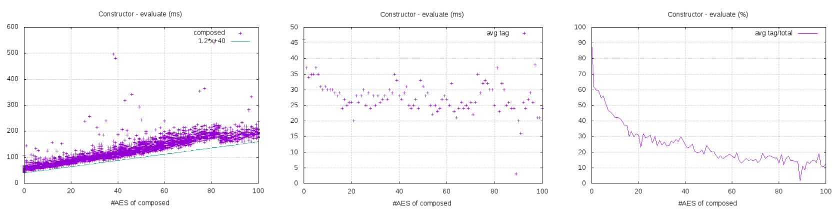
it looks like it is going towards a constant around 20 seconds. This is because the framework is optimized to work on big circuits. Loking at the plots for the average time in figure 3b we see that all the plots are belowe 200 ms, and maybe the average is decreasing slightly. But in worst case it is a constant less than the 200 ms.

This results in a fraction of time used on the circuit that is in worst case is constant, bu looking at the figure 3c it looks like it might be decreasing. But one again we see that many parts of the graph is belove 0 which tells us that the time used on the tag part is small compared to the time used on the total composed circuit. We see that the fraction is less then 1% of the total.

Once again the two parties are similar when looking at the plots. This time again we see that the time used on the composed circuit is linear encreasing in the evaluation phase in figure 4a. When we look at the average time used on the tag in figure 4b we see that it looks like it is linear and slightly decreasing. This results in the fraction we see in figure 4c. Here it is clearly that the fraction of time used on the tag part is decreasing and it looks like it is approximating a constant that is around 10% of the composed circuit.

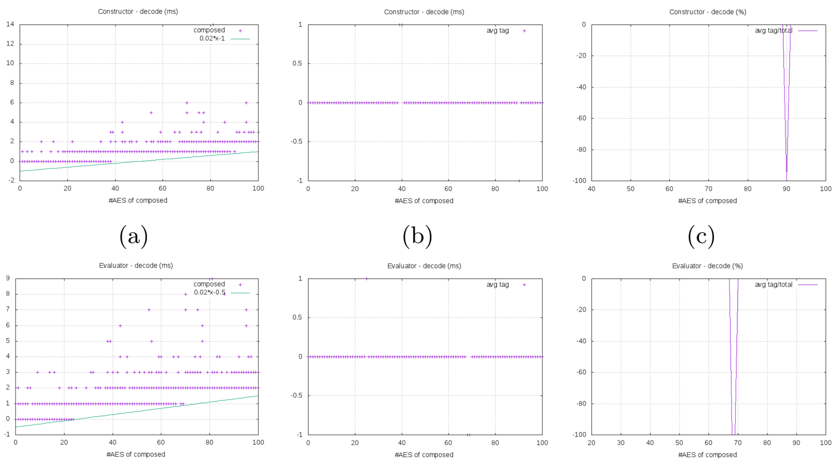
TODO: Describe the decode phase

In the decode phase we do not see the exact same resulta at both parties. But



(a) (b) (c)

Figure 4: The evaluation phase



(a) (b) (c)

(d) (e) (f)

Figure 5: The decode phase

in general it looks like the plots in both 5a and 5d are linear encreasing. From 5b and 5e we see that the average time used on the tag part is constant and negligable. The same goes for the fraction, as seen in 5c and 5f.

TODO: Describe the overall results, total

Summing this up to calculate a fraction of the time used on the tag pat of the

composed circuit dependant on the number of AES circuits. We get the firure 6 which shows that as an upper bound that the tag part of the composed circuit is approcing 0 as the amount of AES circuits encreases.

2.3 Discussion

TODO: Using laptop
TODO: Problems with this way of timing
TODO: Optimization on statistics

3 Conclution

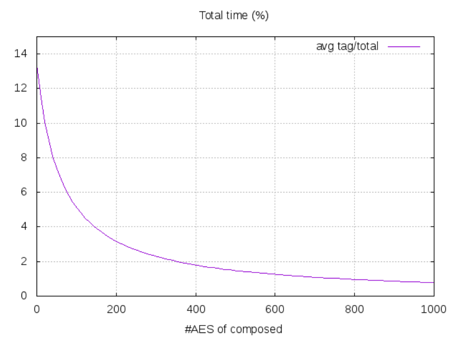


Figure 6: Total tag fraction of composed, $\frac{300}{36x+22500}$

TODO: What was the hypothesis
TODO: What was the conclution

4 Continiued work

TODO: Are the more work to be done?

References

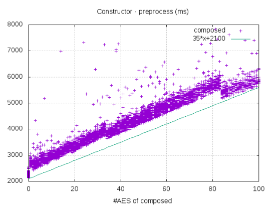
[JNO16] Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. Cryptology ePrint Archive, Report 2016/037, 2016. `http://eprint.iacr.org/2016/037`.

A Data

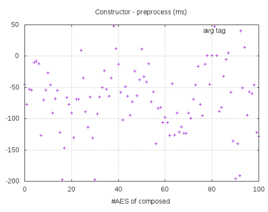
All data generated and used in this report can be found at github[4]. The data was generated using a framework 'duplo' which is an ongoing phd. project by Roberto Trifiletti, which is not public at the moment of writing. All data was collected via a localhost on a standard laptop with a i5-4210U, 1.7GHz processor, 12Gb 1600MHz ram and 6.0Gb/s ssd running Ubuntu 16.10.

The framewok conist of 6 different phases, setup, preprocess, solder, prepare, evaluate and decode. Each of these except setup timed are plottet in the following figures. Each of the subsections contins six polots, the first three are form the constructor and the next three are from the evaluator.

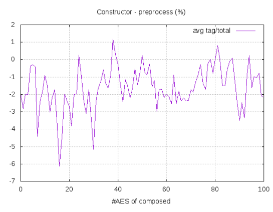
For conclution on the different plots and the hypothesis can be read in the abstract and conclution.



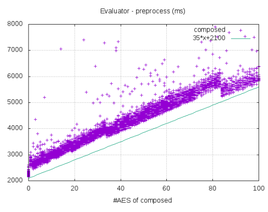
(a) Constructor, total time used on composed circuit



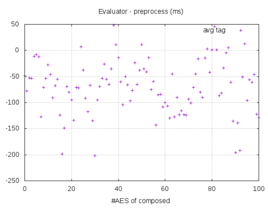
(b) Constructor, average the time used on tag part



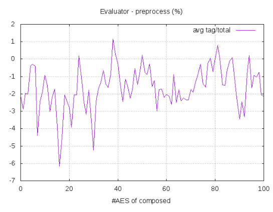
(c) Constructor, fraction of time used on tag part



(d) Evaluator, total time used on the composed circuit



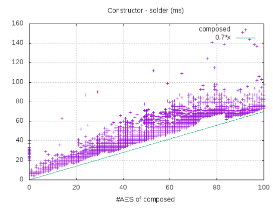
(e) Evaluator, average time used on tag part



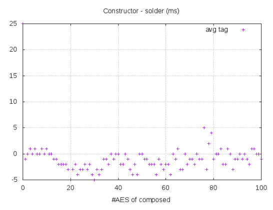
(f) Evaluator, fraction of time used on tag part

Figure 7: The preprocess phase is run on one instance of the ID- and TAG-circuit, while it is run on multiple instances of the AES-circuit.

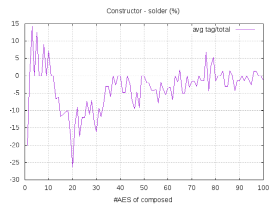
[4] All the data files and plots can be found here: https://github.com/cbobach/mpc-outsourcing



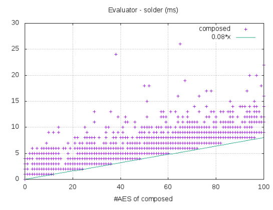
(a) Constructor, total time used on composed circuit



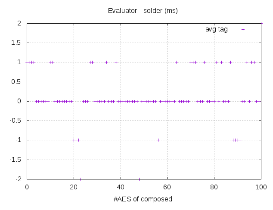
(b) Constructor, average time used on tag part



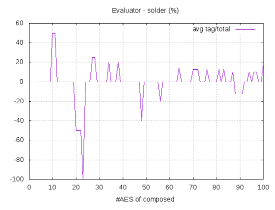
(c) Constructor, fraction of time used on tag part



(d) Evaluator, total time used on the composed circuit

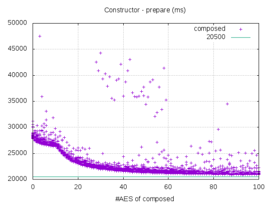


(e) Evaluator, average time used on tag part

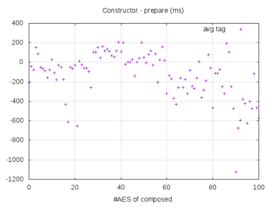


(f) Evaluator, fraction of time used on tag part

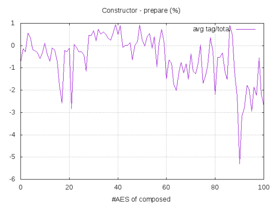
Figure 8: The solder phase is where the preprocessed circuits are composed into one circuit.



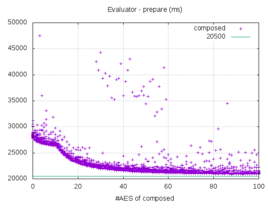
(a) Constructor, total time used on composed circuit



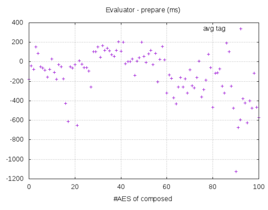
(b) Constructor, average time used on tag part



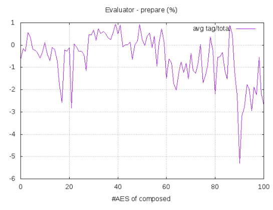
(c) Constructor, fraction of time used on tag part



(d) Evaluator, total time used on the composed circuit

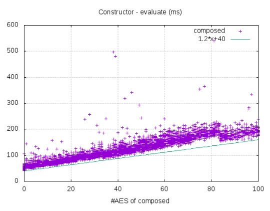


(e) Evaluator, average time used on tag part

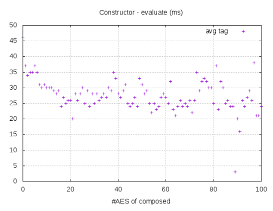


(f) Evaluator, fraction of time used on tag part

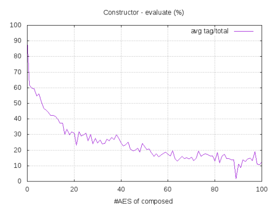
Figure 9: In the prepare phase the composed circuit is prepared for evaluation.



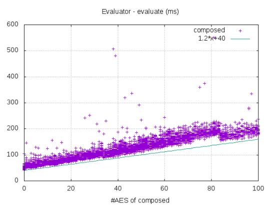
(a) Constructor, total time used on composed circuit



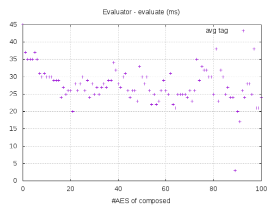
(b) Constructor, average the time used on tag part



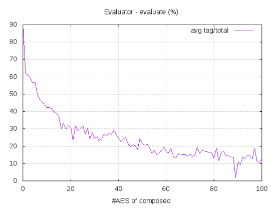
(c) Constructor, fraction of time used on tag part



(d) Evaluator, total time used on the composed circuit

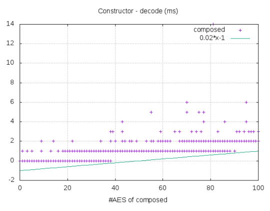


(e) Evaluator, average time used on tag part

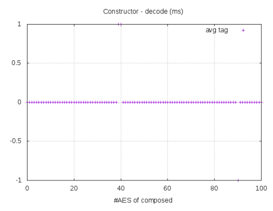


(f) Evaluator, fraction of time used on tag part

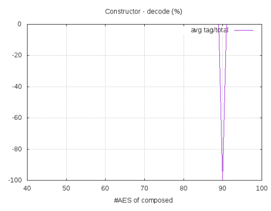
Figure 10: In the evaluation phase the composed circuit is evaluated.



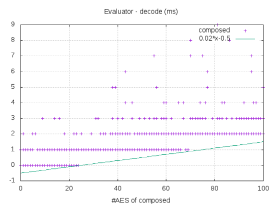
(a) Constructor, total time used on composed circuit



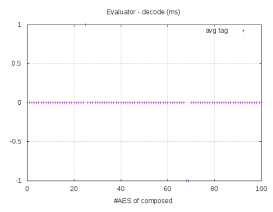
(b) Constructor, average the time used on tag part



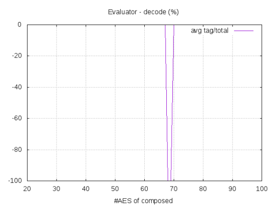
(c) Constructor, fraction of time used on tag part



(d) Evaluator, total time used on the composed circuit



(e) Evaluator, average time used on tag part



(f) Evaluator, fraction of time used on tag part

Figure 11: In the decode phase the output from the composed circuit is decoded.

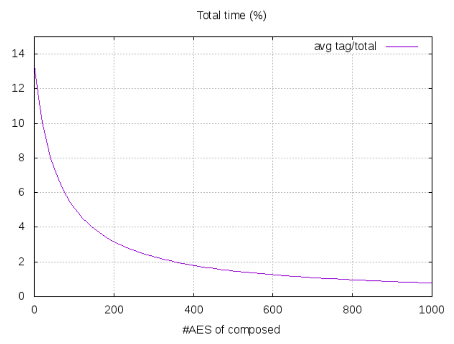


Figure 12: For the plot every line in the composed timings has been summed and a constant upper bound for all the average plots has been used to calculate the fraction. Resulting in $\frac{300}{36x+22500}$ as an upperbound on the tag fraction of the composed circuit.