POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

An Analysis of Random Probing Security Properties in Masked Cryptographic Circuits

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Giuseppe Manzoni, 970636

Advisor:
Prof. Vittorio Zaccaria

Academic year:
2021-2022

Abstract: In this thesis, we explore the random probing security properties of a circuit, i.e. how its security holds up to an adversary able to see the value of each wire with a given probability. In particular, we focus on the Random Probing Security (RPS) and the Random Probing Composability (RPC) of a circuit or gadget. For the RPS, we explore various approximations, while for the RPC we extend the existing definition into a class of RPC-like definitions with three shared properties. In addition to the original definition, we provide two other significant examples of RPC-like properties. Both for RPS and RPC we show how definitions and approximations already present in the literature form meaningful expressions when written using the correlation matrix of the circuit. From this, we show what kind of information about the gadget each of them uses. Lastly, we have created a software tool to calculate the various functions using the correlation matrix of a given generic gadget. We use this tool on various gadgets to analyze the execution time and accuracy trade-off, and to compare those results with existing tools.

Key-words: Side-channel security, Random Probing Security, Automatic verification, Composition, Walsh transform, Correlation matrix

1. Introduction

Side-channel attacks are among the threats against cryptographic implementations that are not covered by the more classical black-box model. This is because they provide the attacker with additional information on the system's state through various means, like power consumption and electromagnetic radiation. The acquisition and analysis of those two leakages is cheap [5], quick [5] and have been fully automated [12] which have greatly lowered the technical knowledge and time required to perform them.
The most common countermeasures against electromagnetic leaks are based on Boolean masking (and the similar additive masking), which was used in the seminal paper [11] to provide a general circuit compiler. This compiler is a function that given any circuit it provides a different one with the same logical behavior but such that it's considered secure in the considered model. As that paper presents two models, it provides a compiler for each. The main and most successful model (henceforth d-threshold probing model) assumes that the attacker, every clock cycle, can use what it has learned to choose the input values and d wires to measure, and then obtain their values once the circuit has stabilized. Later models lift this last requirement (i.e. assuming no transitory periods), but it's still present in the random probing security. To secure a circuit against this attacker [11] encodes the value of each wire with m wires such that their xor (the $\mathbb{F}_2$ addition) has the same value as

the original wire, and then they provide how to substitute the 'not' and 'and' logic gates with gadgets (fixed sub-circuits).
Yet the d-threshold probing model can not represent many physical scenarios, which require a less abstract model like the δ-M-noisy models where each wire of the circuit leaks information with a noise of δ as measured by a metric M. After various attempts at linking the two models, the paper [10] proves that the security of a circuit in the d-threshold model implies its security in a δ-SD-noisy model (SD for Statistical Distance) for a certain d and δ without needing unrealistic leak-free gates. [14] further refines this by tightening the results, but in both those papers we can see how increasing the d, and thus the number of shares, causes the proof to require more and more noise to keep the circuit safe. This is because otherwise they'd be vulnerable to Horizontal attacks, which use multiple wires holding the same or related values to decrease the noise and thus reveal the internal state. The reason why the required noise increases with d can be seen also by noticing how the multiplication gadgets require $(d+1)^2$ internal wires to calculate the initial share-wise products, while they use (unless designed to resist this) only d randoms on each input. This intuitively means that the additional wires can be used to decrease the noise more than it is increased by the additional randoms, and the d-threshold probing model has no way to see this, so the reduction must take it into account instead. For a more formal description, see papers like [4].
At the time of writing, the most common way to model the horizontal attacks is the p-random probing model. The main feature of this model is that every wire of the circuit leaks its exact value independently and with probability p. This is presented as the second model of the [11] paper, and both [10] and [14] use it in the reductions from δ-M-noisy leakage to d-threshold probing. The latter paper also proves that the p-random probing model is equivalent to the δ-SD-noisy model and the δ-ARE-noisy model (ARE for Average Relative Error). The research in this field is mostly aimed at providing compilers that can guarantee any level of security while tolerating leakage up to a constant p, independent of the desired security guarantee. The first paper to provide a solution is [1], and it's followed by a simpler, but still involved, derivation of the same results in [3]. A further improvement that removes the need for expander graphs and algebraic geometric codes is [2] which uses a recursive amplification technique that improves the security by composing a gadget with itself. This technique is further explored and formalized by [5], [7] and [8], the first of which provides a new tool (VRAPS) to test the properties of any given gadget. [9] provides an alternative way to calculate the RPS by composing sub-gadgets, and then [6] gives a concise definition of existing properties while providing a new tool (IronMask) to test those properties for a large subset of all possible gadgets.

Our contribution While the research focuses on more advanced properties like the RPE (Random Probing Expandibility), various questions about the RPS (Random Probing Security) and the RPC (Random Probing Composability) are still open. To this end, our main contributions are the following:

- We provide 4 new approximations of the RPS in different points of the accuracy and time trade-off.
- We show that while [6] introduces a new definition of RPS without comparing it to the existing one in [5], their definition is a sufficient but not necessary condition for the latter. This also means it can be seen as an approximation of the latter.
- We also show with graphs that the approximation of the RPS presented in [5], and currently implemented in the tools, can see concrete improvements, particularly for copy gadgets and multiplications.
- We extend the definition of RPC by creating a class of RPC-like properties that satisfy the essential requirements: RPC is preserved while composing in parallel and in series, and RPC implies RPS.
- We introduce a more accurate RPC-like definition, which means that any non-expanding compiler can now guarantee a higher security with no change.
- We introduce an RPC-like property that is hard to translate in simple terms of simulatability and probability distributions, showing the possibilities to be much wider than what currently researched.
- We introduce three approximations of the new RPC-like properties, in different points of the accuracy and execution time trade-off.
- We express nearly all the mentioned definitions and approximations of RPS and RPC in terms of the correlation matrix to compare them both in accuracy and in structure.
- We show with a numeric example and a simple gadget what seems to be a more generic flaw in IronMask, as it doesn't seem to consider dependencies on the input that arise from combining multiple wires.

This thesis is structured as follows:
Section 1 this introduction.
Section 2 we detail the relevant parts of the state of the art and of the questions that are still open.
Section 3.2 we describe the various approximations of the RPS and compare them using correlation matrices.
Section 3.3 we provide the definition of RPC-like, and we define and compare the various RPC-like properties.
Section 4 we show the practical differences between the various functions and the existing tools.
Section 5 we provide a few conclusive thoughts on the overall work.

2. State of the art and open questions

Since the seminal paper [11], a countermeasure is defined in terms of a circuit compiler. This compiler is composed by three functions. The encoding function Enc that given an input returns its encoding. For the random probing security, Enc is most often additive or Boolean. The decoding function which reverses the encoding, and the function that actually converts the input circuit into one that satisfies the security properties and has the same end-to-end behavior. In the field of random probing security, the compilers usually tend to work by substituting each wire with a set of wires that carry the encoded value, and each logic gate with a gadget that operates directly on the encoding without needing to decode them.
[2] introduces the concept of composite compiler, which is a compiler obtained by compiling multiple times the input circuit. An improvement [8] shows how it's best to use different compilers in different phases. This is because the gadgets a compiler uses to substitute the logic gates need different properties at different steps of the overall compilation. For example, the gadgets used in the last compilation affect the maximum leakage of the wire p that the overall compiler supports, and small gadgets are more suited to this. On the other hand, the asymptotic circuit size expressed in terms of the security parameter improves by using large gadgets.
The paper [2], among other things, also proves how for every p exists a finite basis that can be used to build a compiler tolerating a wire leakage of p, and how every base has a p it won't tolerate. For the binary circuits, it proves how they can't support $p > 0.8$, which is quite far from the current positive results. It's worth noting that we'll show graphs of circuits that tolerate any p, but they are particular cases and not generic compilers.

2.1. State of the art

We follow the structure of circuits given in [5] where every wire can connect a single output with a single input, leakages can happen on the input of the gates (due to the 'only computation leaks' assumption), and copy gates are used to duplicate the value of a wire. To obtain which wires of a gadget g are leaking, we similarly use a probabilistic function $\text{GetLeakingWires}_p^g$. We also define $\text{SD}[A;B]$ as the statistical distance between the distributions of the random variables A and B, while $A \stackrel{d}{=} B$ means that the two have the same probability distribution. We also use $[0^n]$ for a vector of n 0s, and $[1^n]$ for n 1s.
One difference is that we define a gadget g as a probabilistic function $g : \mathbb{F}_2^{I^g} \to \mathbb{F}_2^{O^g+W^g}$ with I^g input bits, O^g output bits, W^g wires that can be leaked and d^g shares for each unmasked input. Additionally, it must be possible to make g deterministic by adding a parameter that is drawn uniformly from a binary vector of size R^g. We also indicate with '$g(x)_w \cap \text{GetLeakingWires}_p^g$' the expression that probabilistically returns the value of the wires being leaked and ignores the output.

RPS (Following [5]) Given a gadget g that encodes the unmasked gate U^g, given $p \in [0,1]$ that is the probability that a single wire leaks, given $\varepsilon \in [0,1]$ that is a limit on how much information on the secret can be leaked, we say that g is (p,ε)-RPS (Random Probing Secure) if there exists a probabilistic simulator Sim^g such that for every unmasked input x, the statistical distance between the sampled wires of the circuit and those of the simulator must be upper bounded by ε. Formally,

$$\forall x \in \mathbb{F}_2^{I^{U^g}}, \quad \varepsilon \geq \text{SD}\left[Sim^g; \underset{W=\text{GetLeakingWires}_p^g}{\text{let}} (W, g(\text{Enc}^g(x))_w \cap W)\right] \tag{1}$$

RPC (Following [5]) Given a gadget g that encodes the unmasked gate U^g, given $p \in [0,1]$ that is the probability that a single wire leaks, given $\varepsilon \in [0,1]$ that is a limit on the probability of 'failure', given a set $S \subseteq \mathbb{F}_2^{d^g}$ that represents which combination of shares is considered safe to leak, and such that $[0^{d^g}] \in S \wedge [1^{d^g}] \notin S$, we say that gadget g is (p,ε,S)-RPC (Random Probing Composable) if a safe combination of leaked outputs can fail to translate to a safe combination of leaked inputs with at most probability ε. This can be formally written using two conditions. The first is that there is a function $in : \mathbb{F}_2^{O^g} \times \mathbb{F}_2^{W^g} \to \mathbb{F}_2^{I^g}$ such that for all safe combinations of leaked outputs O, the probability of in having to select an unsafe combination of inputs is upper bounded by ε. Formally,

$$\forall O \in \mathbb{F}_2^{O^g} : \text{Safe}_S^g(O), \quad \varepsilon \geq \Pr\left[\neg\text{Safe}_S^g(in(O, \text{GetLeakingWires}_p^g))\right] \tag{2}$$

where $\text{Safe}_S^g(O)$ means all unmasked bits represented in O are safe (for every bit, the combination of their selected/leaked shares in S). In addition to this condition, there must be a probabilistic simulator Sim^g such that for every input x of the gadget (not for every unmasked input as with RPS) the inputs chosen by in can be used to perfectly simulate the selected outputs and the leaking wires. Formally, $x \in \mathbb{F}_2^{I^g}, O \in \mathbb{F}_2^{O^g} : \text{Safe}_S^g(O)$:

$$\underset{W=\text{GetLeakingWires}_p^g}{\text{let}} (W, \text{Sim}^g(I, x \cap \text{in}(O,W), O, W)) \stackrel{d}{=} \underset{W=\text{GetLeakingWires}_p^g}{\text{let}} (W, g(x) \cap (O,W)) \tag{3}$$

This is the same as the informal description, as Sim can return the exact same distribution iff its inputs contain all the input dependencies. Overall, this means that given a safe combination of leaked outputs, the input combination needed to simulate them and the leaking wires is an unsafe combination with probability lower or equal to ε.
We do not use the exact definition as [5] as we believe using S and $\text{Safe}_S^g(O)$ is better, and the original behavior can be restored by using:

$$S^t = \{w \mid w \in \mathbb{F}_2^{d^g} \wedge |w| \leq t\}$$

Where $|w|$ of a binary vector w always indicates the number of 1s in w, as if it was a set. Our S was taken as a more generic version of their 'adequate subsets of $[n^k]$' that they use in their definition of 'RPE with $\{S_k\}_{k\in\mathbb{N}}$'. Our reasons for using the S in the RPC are:

- It's more clear as it abstracts out the implementation detail of how the safe set is defined.
- It's more general with little to no downsides, as it has all the properties necessary to prove the relevant theorems.
- Their definition of RPC causes a formal flaw in one of their proofs about the RPE. Their intended way of proving the security of the expanding compiler is to use the RPE property to create gadgets with the required security level, then prove that the result of the expansion is RPC and finally compose the gadgets using the RPC property to form the whole circuit. Except that while they prove that RPE implies RPC, they use a more generic RPE (called 'RPE with $\{S_k\}_{k\in\mathbb{N}}$') to prove the expansion. That set S is not in the form supported by their definition of RPC (which is the S^t) but it's a fractal set defined recursively as each layer of expansion is applied. We can't see how such a sophisticated set can fit in the flat S^t, and defining RPC in this way would solve the problem.

Still, the difference is just a formal fix, and we don't believe this to be worth being considered a separate definition, even if this does make it more generic.

2.2. Open questions

This leaves many open directions of research, some of which answered by this paper, while others still open. Among the following we find: the paper [5] gives the definition of RPS, which is immediately followed by an approximation that is used in the rest of the paper. Can a more accurate approximation be found? Or a faster one? What kind of information about the gadget do these approximations use? How does the RPS definition given by IronMask [6] relate to the one provided by the previous paper and reported here? (Section 3) And how much does the accuracy vary? (Section 4)
As per the RPC, can we generalize it to a wider rage of alternative definitions? Are there definitions that guarantee a higher f for the RPS property of the resulting gadget? Can we to define an RPC-like property using the SD like for the RPS? Is there a definition that is hard to express with the simulatibility? How do the expressions of alternative definitions relate, and are they linked in some way to the approximations of RPS? (Section 3)
As those definitions and approximations use probability distributions, does writing any of them using the correlation matrix provide any insightful information on how they work? Or which information they use? (Section 3) How do they compare in accuracy and execution time? How does a software tool written following these expressions compare to existing tools? (Section 4)
There was one question we didn't expect asking: are the existing tools reliable? We assumed they were, but after comparing many of IronMask's results with ours, we tried calculating a few simple gadgets by hand, and they show that the source of the discrepancy is in IronMask. A numerical example is reported in Section 4.
We couldn't find positive answers to all questions, for example: is there an alternative definition of the RPC that uses the 'for every underlying input' like the RPS does? Is there a compact way to express the exact definition of RPS? We found an algorithm, but no closed formula to calculate it like we did for every other definition or approximation.
This thesis was written to explore and show that there are good alternatives to the existing definitions and approximations, this while using the same definition of what makes a circuit secure (RPS). Our work creates new questions: is there a definition of RPC that is the most accurate possible? If so, what is the execution time required to calculate exactly its minimal ε? Can a class of alternative definition of RPE be found and be expressed in terms of the correlation matrix, like for the RPC? Can they be based on a more accurate definition of RPC, like the one we introduce using the SD?

3. Remodeling random probing security concepts with correlation matrices

3.1. Introduction

Given a probabilistic vector x we can define mass $[x]$ as the probability mass function of x, and conversely given a probability mass function x we can sample a probabilistic vector using $\leftarrow x$. Then we can define the Fourier transform as the invertible function $\mathcal{F}[f]$ that given a function $f : \mathbb{F}_2^{I^f} \to \Re$ returns a function of the same type:

$$\mathcal{F}[f](\gamma) = \sum_{\alpha \in \mathbb{F}_2^{I^f}} H_{\gamma,\,\alpha} f(\alpha)$$

with $H : \mathbb{F}_2^n \times \mathbb{F}_2^n$ the Hadamard matrix of size n: $H_{\omega,\,\alpha} = (-1)^{\omega^T \alpha}$.
Then for a deterministic vectorial function $f : \mathbb{F}_2^{I^f} \to \mathbb{F}_2^{O^f}$, its correlation matrix $C^f : \mathbb{F}_2^{O^f} \times \mathbb{F}_2^{I^f} \to \Re$ is such that:

$$C^f_{\omega,\,\alpha} = 2^{-I^f} \sum_{x \in \mathbb{F}_2^{I^f}} H_{\omega,\,f(x)} H_{x,\,\alpha}$$

With ω, α that are the spectral coordinates of the correlation matrix.
The properties of the correlation matrix relevant for this thesis can be seen in [13] and are here reported. Given a deterministic vectorial function f and a probability mass function x, we can define y as the probability mass function relative to x's output $y = \text{mass}[f(\leftarrow x)]$, then:

$$\mathcal{F}[y](\omega) = \sum_{\alpha \in \mathbb{F}_2^{I^f}} C^f_{\omega,\,\alpha} \mathcal{F}[x](\alpha)$$

Which means that it can be used to link the input and output probability mass functions:

$$y = \mathcal{F}^{-1}\left[\omega \to \sum_{\alpha \in \mathbb{F}_2^{I^f}} C^f_{\omega,\,\alpha} \mathcal{F}[x](\alpha)\right]$$

Where $\omega \to ...$ indicates a lambda with one parameter. Also, given two deterministic vectorial functions f, g then the parallel composition is:

$$C^{f|g}_{\omega_f|\omega_g,\,\alpha_f|\alpha_g} = C^f_{\omega_f,\,\alpha_f} C^g_{\omega_g,\,\alpha_g}$$

And the series composition (when meaningful) is:

$$C^{f \circ g}_{\omega,\,\alpha} = C^f_{\omega,\,k} C^g_{k,\,\alpha}$$

And those two together can prove the composition of arbitrary functions.
From those properties, we can define the correlation matrix of a gadget g. We can call g' the deterministic function relative to g so that

$$g(x) = g'(x| \leftarrow \mathbb{F}_2^{R^g})$$

as allowed by the definition of gadget. Then we have proved that the correlation matrix $C^g : \mathbb{F}_2^{O^f + W^f} \times \mathbb{F}_2^{I^f} \to \Re$ is:

$$C^g_{\omega,\,\alpha} = C^{g'}_{\omega,\,\alpha|[0^{R^g}]}$$

The properties of the standard correlation matrix reported above also apply for this matrix and the function g, with the serial composition of $g \circ h$ defined so that only the output of h goes into the input of g, and the internal wires of both become the internal wires of the compound gadget.
From the correlation matrix we can define a compact correlation matrix $L^g : \mathbb{F}_2^{O^g} \times \mathbb{F}_2^{I^g} \to \Re^{W^g+1}$, which uses as indexes the spectral coordinate of the input and of the output (but without the internal wires) and returns an array: for each index i, how many combinations of i leaking wires have a correlation between that input and output coordinates. With a formula:

$$(L^g_{\omega\,\alpha})_i = \sum_{w \in \mathbb{F}_2^{W^g} : |w| = i} \text{is}\left[\exists \psi \preceq w,\ C^g_{\omega|\psi,\,\alpha} \neq 0\right]$$

where $\preceq$ is element-wise $\leq$, and if we see a $\mathbb{F}_2^n$ vector as sets of indexes, then it's isomorphic to $\subseteq$. We have proved that the composition in parallel of two gadgets g, h is:

$$(L^{g|h}_{\omega_g|\omega_h\ \alpha_g|\alpha_h})_i = \sum_{j\in\mathbb{Z}_{i+1}} (L^g_{\omega_g\ \alpha_g})_j (L^h_{\omega_h\ \alpha_h})_{i-j}$$

And the composition in series of two gadgets g, h is:

$$(L^{g\circ h}_{\omega\ \alpha})_i \leq \sum_{j\in\mathbb{Z}_{i+1}} \sum_{k\in\mathbb{F}_2^{O^h}} (L^g_{\omega\ k})_j (L^h_{k\ \alpha})_{i-j}$$

3.2. Comparing approximations of the RPS using correlation matrices

In this section, we initially provide an alternative definition of RPS that makes more explicit the influence of the GetLeakingWires^g_p in the expression. From this, we give an upper bound on a sub-expression of the new definition. We also analyze the Walsh transform of the gadget (and of the expression it's used in) to find which rows and columns of its correlation matrix have an influence on the result.
After this preliminary analysis, we introduce several approximations of the RPS properties, and all of which can be seen as sufficient but not necessary conditions, as the minimal ε they find is higher than the minimum required by the RPS. For this reason, to compare the accuracy of two approximation we use the ε, and the lower it is, the more accurate. More formally, we say that an approximation A is more accurate than an approximation B if we have proved both that in all gadgets

$$\forall p \in [0,1], \varepsilon_A(p) \leq \varepsilon_B(p)$$

and if there is a gadget for which:

$$\forall p \in (0,1), \varepsilon_A(p) < \varepsilon_B(p)$$

The first approximation considered is RPS_VRAPS which is an exact translation of the first approximation done in [7] right after giving the definition of RPS. Then we improve it with the upper bound, creating RPS_COR3. As [5] approximates the definition of RPS immediately after giving it, we wondered how much was lost by that approximation, and so we provide RPS_COR1, which we have proved to be equivalent to the RPS definition when we use as simulator the real circuit with random input, which is the same simulator of the upper bound. Due to the high execution times, we provide a further approximation called RPS_COR2.
We then provide an exact translation of the RPS definition given in [6] and we show that it can be seen as an approximation of the RPS definition given in [5] and reported here. Henceforth, we call that expression RPS_IRONMASK from the paper and the tool that calculates it.
Lastly we provide RPS_L, a further approximation of RPS_COR3 which is not based on the correlation matrix, but on the reduced correlation matrix introduced at the end of 3.1. As the original intention was to base everything directly on the correlation matrix to have a level field to compare the approximations on, this kind of approximation was only found accidentally while searching for an alternative RPE toward the end of the thesis. The RPE part was not included as it wasn't completed in time, but the RPS_L seems promising due to the lower complexity. Still, it was not implemented in our tool as it works quite differently, and it would take time to optimize what would amount to a new tool to a comparable level, which is required to have a fair comparison.

RPS Before confronting the alternative approximations of the RPS and their trade-offs, it's useful to make explicit the effects that the leaking wires have in the definition of RPS itself: A gadget g is (p,ε)-RPS iff there exists a probabilistic simulator $SimW^g$ such that:

$$\varepsilon \geq \underbrace{\max_{x\in\mathbb{F}_2^{I^{Ug}}}}_{\text{all unmasked in}} \underbrace{\sum_{W\in\mathbb{F}_2^{W^g}} \overbrace{p^{|W|}(1-p)^{W^g-|W|}}^{\text{prob. } W \text{ is the leakage}}}_{\text{weighted average}} \underbrace{\text{SD}\left[\overbrace{SimW^g(W)}^{\text{simulator}}; \overbrace{g(\text{Enc}^g(x))_w \cap W}^{\text{real}}\right]}_{\text{SD with leakage fixed to } W} \tag{4}$$

As it's used often in this section, we give a name to the SD of Expression (4). In this way, we can define the lower bound on the ε only by specifying this part of Expression (4):

$$SD^{g,W,x}_{rps} = \underbrace{\text{SD}\left[\overbrace{SimW^g(W)}^{\text{simulator}}; \overbrace{g(\text{Enc}^g(x))_w \cap W}^{\text{real}}\right]}_{\text{SD with leakage fixed to } W} \tag{5}$$

We can find an upper bound on Expression (4) by choosing as simulator the real gadget and giving it a uniformly sampled input:

$$SimW^g(W) = g(\mathrm{Enc}^g(\leftarrow \mathbb{F}_2^{I^{U^g}}))_w \cap W$$

Then we can prove for that $SimW$ that:

$$\forall g, W, x, \quad SD_{rps}^{g,W,x} \leq \underbrace{1 - 2^{-I^{U^g}}}_{\text{upper bound}} \tag{6}$$

In practice this means that given a SimW we can create a SimW' that can chose (based on W) to either return the original $SimW(W)$ or return the simulator used in Expression (6) which would allow us to guarantee that upper bound on all SimW'.

Effects of encoding Before going forward, it's useful to define the correlation matrix of the encoding for a given gadget g:

$$C^{\mathrm{Enc}^g}_{\omega,\,\alpha} = \mathrm{is}\,[\omega = \mathrm{inShares}^g(\alpha)]$$

Where is [] gets an expression as a parameter and returns its truth value as a $\mathbb{F}_2$ Boolean. $\mathrm{inShares}^g(i)$ returns the set of shares relative to chosen underlying input i. Formally:

$$i \in \mathbb{Z}_{I^{U^g}}: \quad \mathrm{inShares}^g(i) = [0^{id^g}]|[1^{d^g}]|[0^{(I^{U^g}-(i+1))d^g}]$$

$$i \in \mathbb{F}_2^{I^{U^g}}: \quad \mathrm{inShares}^g(i) = \sum_{j:i_j=1} \mathrm{inShares}^g(j)$$

In concrete, for a gadget with 2 inputs and 3 shares:

$i =$	$\mathrm{inShares}^g(i) =$
$[0,0]^T$	$[0,0,0,0,0,0]^T$
$[1,0]^T$	$[1,1,1,0,0,0]^T$
$[0,1]^T$	$[0,0,0,1,1,1]^T$
$[1,1]^T$	$[1,1,1,1,1,1]^T$

This means the $C^{\mathrm{Enc}^g}_{\mathrm{row},\,\mathrm{column}}$ of a gadget with 2 inputs and 3 shares is:

	$[0,0]^T$	$[1,0]^T$	$[0,1]^T$	$[1,1]^T$
$[0,0,0,0,0,0]^T$	1	0	0	0
$[1,1,1,0,0,0]^T$	0	1	0	0
$[0,0,0,1,1,1]^T$	0	0	1	0
$[1,1,1,1,1,1]^T$	0	0	0	1
otherwise	0	0	0	0

As the RPS definition uses $g \circ \mathrm{Enc}^g$, we can use the composition of correlation matrices to see how of the only relevant columns α of C^g are those that have a 1 in the row α of C^{Enc^g}, as the others are all multiplied by 0. Additionally, we know that given the input probability mass function x:

$$\mathcal{F}[x]([0^{I^x}]) = \sum_{\alpha \in \mathbb{F}_2^{I^x}} x(\alpha) = 1$$

Which means we can ignore that column of the correlation matrix as it doesn't reveal anything on the input value.
In practice, this means that the RPS considers at most the components:

$$\underbrace{\forall \alpha \in \mathbb{F}_2^{I^{U^g}} : \alpha \neq [0^{I^{U^g}}]}_{\text{at least one unmasked in}} \quad \overbrace{\forall \omega \in \mathbb{F}_2^{O^g+W^g}}^{\text{any out and internal wire}} \quad C^g_{\omega,\,\underbrace{\mathrm{inShares}^g(\alpha)}_{\substack{\text{all shares of } i \\ \text{are } =\alpha_i}}} \tag{7}$$

Effects of selecting leaking wires As the RPS definition uses only the subset W of all the wires, we can define the helper function

$$\text{select}^W(x) = x \cap W$$

such that

$$(\text{select}^W \circ g_w)(x) = g_w(x) \cap W$$

The $C^{\text{select}^W}_{\text{row, column}}$ for $W = [1, 0, 1, 0]^T$:

	$[0,0,0,0]^T$	$[1,0,0,0]^T$	$[0,0,1,0]^T$	$[1,0,1,0]^T$	otherwise
$[0,*,0,*]^T$	1	0	0	0	0
$[1,*,0,*]^T$	0	1	0	0	0
$[0,*,1,*]^T$	0	0	1	0	0
$[1,*,1,*]^T$	0	0	0	1	0

Where * is used to mean that it covers both 0 and 1. We can see that any column ω whose spectral coordinate contains a bit not in W has all components to 0 (the 'otherwise' column). As the RPS uses $\text{select}^W \circ g_2 \circ \text{Enc}^g$, this means that the corresponding rows ω in the $C^{g_2}_{\omega,\,\alpha}$ are not considered, as they're multiplied by 0. This with Expression (7) means for a given W the considered components are at most

$$\underbrace{\forall \alpha \in \mathbb{F}_2^{I^{Ug}} : \alpha \neq [0^{I^{Ug}}]}_{\text{at least one unmasked in}} \quad \overbrace{\forall \psi \preceq W}^{\text{any subset of } W} \quad C^g_{\underbrace{[0^{O^g}]}_{\text{no out}} \,|\psi,\, \underbrace{\text{inShares}^g(\alpha)}_{\substack{\text{all shares of } i \\ \text{are } =\alpha_i}}} \tag{8}$$

RPS_VRAPS The ideal would be to use the best SimW possible, but it's involved to do so and even the original paper [5] uses a simulator with failure that either returns a perfect simulation ($SD = 0$) or fails and assumes the worst ($SD = 1$). We have proved that with this $SimW^g$:

$$SD^{g,W,x}_{rps} = \text{is} \left[\underbrace{\exists \alpha \in \mathbb{F}_2^{I^{Ug}} : \alpha \neq [0^{I^{Ug}}]}_{\text{at least one unmasked in}} \quad \overbrace{\exists \psi \preceq W,}^{\text{any subset of } W} \quad \overbrace{C^g_{\underbrace{[0^{O^g}]}_{noout} \,|\psi,\, \underbrace{\text{inShares}^g(\alpha)}_{\substack{\text{all shares of } i \\ \text{are } =\alpha_i}}} \neq 0}^{\text{any correlation with}} \right]$$

Which means that the SD is 0 iff all the components of Expression (8) are 0, i.e. there is no correlation between the W and the secret input.
From this expression we can obtain an asymptotic execution time to calculate the first $c < W/2$ coefficients, with W the same as W^g but with no multiplicity due to copy gates:

$$O\left(\underbrace{\binom{W^g}{c}}_{\text{from } W \text{ to } W^g} + \underbrace{\overbrace{\binom{W}{c}}^{\substack{\text{over all} \\ \text{selected rows}}} \left(\underbrace{2^c}_{\substack{\text{num of} \\ \text{subrows}}} + \underbrace{\overbrace{2^{I^g} I^g}^{\substack{\text{row-only} \\ \text{calculations}}}}_{\substack{\text{row} \\ \text{tansform}}} \right)}^{\substack{\text{merge} \\ \text{subrows}}}}_{\text{calculations over } W} \right)$$

All the complexities calculated by our tool can be expressed in this form, and only expresses that we can calculate and store the inner expressions to avoid repeating the same calculations. These complexities also ignore other optimizations that depend on the gates and wiring of the parameter gadget.
The complexity to calculate all the coefficients is:

$$O\left(2^{W^g} + 2^W(2^W + 2^{I^g} I^g)\right)$$

RPS_COR3 Using Expression (6) we can improve RPS_VRAPS as its SD only depends on W. This results in:

$$SD^{g,W,x}_{rps} = \underbrace{(1 - 2^{-I^{Ug}})}_{\text{upper bound}} \text{is} \underbrace{\left[\underbrace{\exists \alpha \in \mathbb{F}_2^{I^{Ug}} : \alpha \neq [0^{I^{Ug}}]}_{\text{at least one unmasked in}} \quad \overbrace{\exists \psi \preceq W,}^{\text{any subset of } W} \quad \overbrace{C^g_{\underbrace{[0^{O^g}]}_{noout} \,|\psi,\, \underbrace{\text{inShares}^g(\alpha)}_{\substack{\text{all shares of } i \\ \text{are } =\alpha_i}}} \neq 0}^{\text{any correlation with}} \right]}_{\text{the same of RPS\_VRAPS}}$$

This has, of course, the same asymptotic complexity as RPS_VRAPS.

RPS_COR1 A better approximation would be to directly use the SimW used in Expression (6):

$$SimW^g(W) = g(\mathrm{Enc}^g(\leftarrow \mathbb{F}_2^{I^g}))_w \cap W)$$

We have proved that with this $SimW^g$:

$$SD^{g,W,x}_{rps} = \overbrace{2^{-|W|}}^{\text{from } \mathcal{F}^{-1}[\mathcal{F}[.]]} \underbrace{\frac{1}{2} \sum_{w \preceq W}}_{\text{the SD's}} \left| \underbrace{\sum_{\alpha \in \mathbb{F}_2^{I^{U^g}} : \alpha \neq [0^{I^{U^g}}]}}_{\text{at least one unmasked in}} \overbrace{\sum_{\psi \preceq W}}^{\text{all subset of } W} C^g_{\underbrace{[0^{O^g}]}_{noout} | \psi, \underbrace{\text{inShares}^g(\alpha)}_{\substack{\text{all shares of } i \\ \text{are } = \alpha_i}}} \overbrace{H_{w,\, \psi} H_{\alpha,\, x}}^{\text{from } \mathcal{F}^{-1}[\mathcal{F}[.]]} \right|$$

We can see how this expression uses the exact same components of RPS_COR3 and RPS_VRAPS, but instead of checking only if those components are not all = 0, it also uses both their value and their position in the calculation of the SD, so one would expect a higher accuracy (which can easily proven to be the case).
From this expression we can see the asymptotic execution time to calculate the first $c < W/2$ coefficients, with W the same as W^g but with no multiplicity due to copy gates:

$$O\left(\binom{W^g}{c} 2^{I^{U^g}} + \binom{W}{c}(2^{I^{U^g}+2c} + 2^{2I^{U^g}} + 2^{I^g} I^g)\right)$$

While to calculate all coefficients we have (assuming all inputs are used at least once internally):

$$O\left(2^{W^g + I^{U^g}} + 2^{3W + I^{U^g}}\right)$$

RPS_COR2 A different approximation that can derived from RPS_COR1 and Expression (6) by moving the $|\cdots|$ inward:

$$SD^{g,W,x}_{rps} = min\left(\underbrace{1 - 2^{-I^{U^g}}}_{\text{upper bound}}, \underbrace{\frac{1}{2}}_{\text{from SD}} \underbrace{\sum_{\alpha \in \mathbb{F}_2^{I^{U^g}} : \alpha \neq [0^{I^{U^g}}]}}_{\text{at least one unmasked in}} \overbrace{\sum_{\psi \preceq W}}^{\text{all subset of } W} \left| C^g_{\underbrace{[0^{O^g}]}_{noout} | \psi, \underbrace{\text{inShares}^g(\alpha)}_{\substack{\text{all shares of } i \\ \text{are } = \alpha_i}}} \right| \right)$$

This, of course, uses the exact same components as the others, but compared to RPS_COR1 it only uses the module of values while ignoring their position. It also has a single sum over the subset of leaking wires, and it doesn't vary with x. We have proved that it has an accuracy that is in between RPS_COR1 and RPS_COR3, hence the name.
From this expression, we can obtain the same asymptotic execution time as RPS_COR3, but the optimizations usually make that one faster, see Section 4.2.

RPS_IRONMASK We now provide an equivalent expression to the definition of RPS given in [6], but we have also proved that this expression is an approximation of the RPS definition given here, and when considered this way it's less accurate than RPS_VRAPS.

$$SD^{g,W,x}_{rps} = \text{is} \left[\underbrace{\exists i \in \mathbb{Z}_{I^{U^g}}}_{\text{any unmasked in}} \overbrace{\forall d \in \mathbb{Z}_{d^g}}^{\text{all its shares}} \underbrace{\exists \alpha \in \mathbb{F}_2^{I^g} : \{\overbrace{d + id^g}^{\substack{\text{share } d \\ \text{input } i}}\} \in \alpha}_{\text{there is } \alpha \text{ containing that share}} \overbrace{\exists \psi \preceq W}^{\text{all subset of } W} \overbrace{C^g_{\underbrace{[0^{O^g}]}_{noout} | \psi,\, \alpha} \neq 0}^{\text{any correlation with}} \right]$$

This, while it uses the same rows, actually uses more columns than the Expression (8) says are required. This is because the original definition in [6] implicitly uses $y \in \mathbb{F}_2^{I^g}$ directly as a parameter of g instead of using $\mathrm{Enc}^g(x), x \in \mathbb{F}_2^{I^{U^g}}$.
This means that as long as an input has all its shares selected, the other inputs can have any combination of shares, not only the all-or-nothing of the other approximations. The results leading to Expression (8) clearly show how these combinations correlate only with the random bits of the encoding, and not with the secret unmasked input, hence the lower accuracy.
From this expression we can see that the asymptotic execution time to calculate the first $c < W/2$ coefficients, with W the same as W^g but with no multiplicity due to copy gates:

$$O\left(\binom{W^g}{c} + \binom{W}{c}(2^{I^g + c} + 2^{2I^g})\right)$$

While to calculate all coefficients it's (assuming all inputs are used at least once internally):

$$O\left(2^{W^g} + 2^{I^g+2W}\right)$$

This means that, at least if calculated this way, the execution time is asymptotically higher than RPS_VRAPS, even if it has lower accuracy.

RPS_L Using the matrix L^g we can further approximate the RPS_COR3 in order to make it faster to calculate all coefficients:

$$\varepsilon \geq \underbrace{\sum_{i \in \mathbb{Z}_{W^g+1}}}_{\substack{\text{over possible} \\ \text{coefficients}}} \overbrace{p^i(1-p)^{W^g-i}}^{\substack{\text{prob. of one} \\ \text{combination with} \\ i \text{ wires is leaking}}} \overbrace{\underbrace{(1-2^{-I^{U^g}})}_{\text{upper bound}} min\left(\underbrace{\begin{pmatrix} \overbrace{W^g}^{\substack{\text{combinations} \\ \text{with } i \text{ wires}}} \\ i \end{pmatrix}}_{\substack{\text{approximated number of leaking combinations of } i \text{ wires.}}}, \underbrace{\sum_{\alpha \in \mathbb{F}_2^{I^{U^g}} : \alpha \neq [0^{I^{U^g}}]}}_{\substack{\text{at least one unmasked in} \\ \text{all shares of } i \\ \text{are } = \alpha_i}} \overbrace{(L^g_{\underbrace{[0^{O^g}]}_{noout}} \underbrace{\text{inShares}^g(\alpha)}_{\substack{\text{all shares of } i \\ \text{are } = \alpha_i}})_i \neq 0}^{\text{compact correlation matrix}}\right)}^{\text{coefficient } c_i \text{ of } \varepsilon(p)}$$

Accuracy-wise this worse, as the upper limit is on the whole coefficient and not on the single combination of wires. Additionally, the $\max_\alpha$ of RPS_COR3 here becomes a $\sum$, and there is an accuracy loss in case L^g is calculated by composition. Yet in this way the asymptotic complexity of a circuit of width w and with l non-copy gates is:

$$O\left(2^w l W^{g2}\right)$$

While the other fastest approximation is:

$$O\left(2^{W^g} + 2^W(2^W + 2^{I^g} I^g)\right)$$

It's hard to compare the two, as the first highly depends upon the width w, which in turn depends on the optimizations done by the tool.
We can consider now a broad class of gadgets. Any meaningful gadget with randoms has $W^g \geq W + 2R^g$ due to the copy gates on the randoms, which means $w \leq W^g - 2R^g + O^g$. Additionally, in nearly all gadgets R^g grows more than logarithmically in the number of gates, to avoid horizontal attacks. This means that if the additional condition of $R^g/O^g > 1/2$ is also satisfied, then this is the asymptotically fastest approximation (ignoring additional optimizations), even with the worst w possible.
A more concrete example is to calculate the complexities for a circular refresh:

$$RPS\_L : O(2^{d^g+2} d^{g3})$$

$$RPS\_COR3 : O(2^{6d^g})$$

making the exponent of this approximation a sixth of the other one.

The following images (Figure 1, Figure 2) show the relationship between the various approximations in terms of accuracy and execution time.

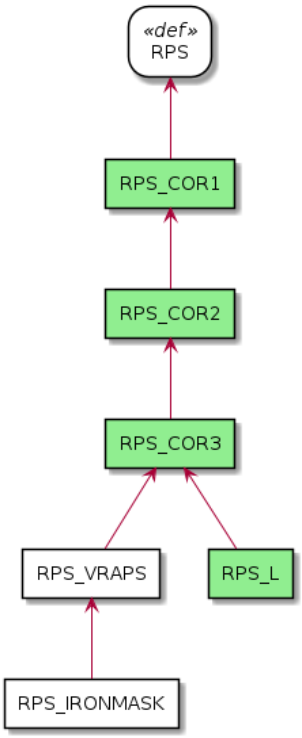


Figure 1: Accuracy, arrows toward the higher accuracy. Light green for the newly introduced approximations.

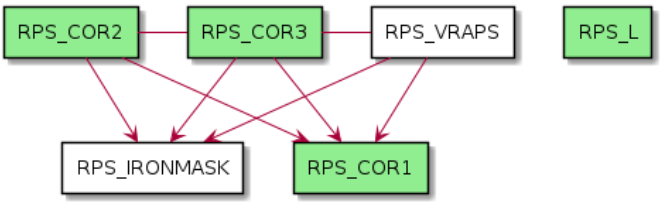


Figure 2: Asymptotic timing, arrows toward the higher complexity. Links if the complexity is the same. Light green for the newly introduced approximations.

3.3. Extending RPC into a class of properties

As explained in the previous sections, the required characteristics of a RPC-like property are three. We consider a generic property K such that given a gadget g, the two values $p, \varepsilon \in [0, 1]$ and a set of shares that can be safely leaked S, we can wonder if g is (p, ε, S)-K. Then we can say that K is RPC-like if all the following are true:

- (p,ε,S)-K implies (p,ε)-RPS
- if given two gadgets g, h respectively (p,ε_g,S)-K and (p,ε_h,S)-K we have that the parallel $g|h$ is (p,ε_g+ε_h,S)-K.
- if given two gadgets g, h respectively (p,ε_g,S)-K and (p,ε_h,S)-K we have that the series $g \circ h$ (when meaningful) is (p,$\varepsilon_g + \varepsilon_h$,S)-K.

From these it's simple to prove that any composition of gadget implies an RPS with a ε that is the sum of the ε of all the gadgets.
For this reason, we can compare the accuracy of two RPC-like properties by comparing the accuracy of the RPS they guarantee.

We report three definitions. The first uses simulations with failure and is the original one of [5] which we call RPC_VRAPS from the name of their tool. Yet those simulations with failure are akin to those of RPS_VRAPS, and so we introduce a definition called RPC_SD that uses the Statistical Distance like the RPS definition, with the aim to improve the accuracy. Lastly, we go in the opposite direction and improve the asymptotic execution speed. As the tool is based on the correlation matrix, we provide a definition RPC_C defined directly in terms of the correlation matrix. This definition also happens to be hard to translate in terms of probability distributions and simulations, which reveals that there are many more avenues of research than those usually considered in the literature.
In addition to those definitions, we provide three approximations, which all have a corresponding approximation in the RPS. The first is RPC_C_L, which is an approximation of RPC_C using the same matrix as RPS_L, and it has the same asymptotic execution time. The other two are both approximations of RPC_SD using the real circuit as simulator and randoms for the missing input bits. This leads us to RPC_SD_COR1 which is similar to RPS_COR1, and to RPC_SD_COR2 which mimics RPS_COR3. Similar to what happens in the

RPS, RPC_SD_COR2 is the same as RPC_VRAPS multiplied by a constant, which is the upper bound on RPC_SD.

RPC_VRAPS It goes without saying that RPC_VRAPS is RPC-like (the proof can be easily adapted from the properties in the original paper [5]), so we directly provide an alternative definition (that we have proved to be equivalent). A given gadget g is (p,ε,S)-RPC_VRAPS iff:

$$\varepsilon \geq \underbrace{\max_{\substack{O\in\mathbb{F}_2^{O^g} \\ :\text{Safe}_S^g(O)}}}_{\text{for all safe } O} \underbrace{\sum_{W\in\mathbb{F}_2^{W^g}} \overbrace{p^{|W|}(1-p)^{W^g-|W|}}^{\text{prob. } W \text{ is the leakage}}}_{\text{weighted average}} \text{is} \underbrace{\left[\overbrace{\forall I \in \mathbb{F}_2^{I^g} : \text{Safe}_S^g(I),}^{\text{All possible results of } in} \underbrace{\exists \alpha \in \mathbb{F}_2^{I^g} : \alpha \not\preceq I}_{\substack{\text{any coord. that} \\ \text{selects bit outside of } I}} \overbrace{\underbrace{\exists \omega \preceq (O|W)}_{\text{With any sub row}}}^{\substack{\text{Due to the} \\ \text{out selection}}} \overbrace{C^g_{\omega,\,\alpha} \neq 0}^{\text{Any correlation}}\right]}_{\text{All possible result of } in \text{ have a dependency they don't cover}}$$

We can recognize various parts. Like for RPS_VRAPS, this is a weighted sum over the probability of W being the exact set of leaking wires of the SD between a simulator and the actual value. In this case the simulator is one that either outputs the perfect distribution or fails as it'd require an unsafe input combination to do the simulation. The use of $= 0$ is because it succeeds ($SD = 0$) only if there is no correlation between the selected inputs and outputs, the $\omega \preceq O$ and the $\psi \preceq W$ that originate from selecting only a subset of the outputs and leaking wires, like in Expression (8). The only new part is the

$$\forall I \in \mathbb{F}_2^{I^g} : \text{Safe}_S^g(I), \exists \alpha \in \mathbb{F}_2^{I^g} : \alpha \not\preceq I, \cdots$$

In words, I is one of the possible results of in (see the definition) that don't cause the simulation to fail (as they're safe), and α proves that it can't be used to generate a perfect distribution as there is some missing correlation that isn't covered by I.
This means that it's not necessary to have a correlation with an unsafe input combination. E.g. for $S^t, t = 3$, the correlation from only the safe input coordinates $[1,1,1,0]^T, [1,1,0,1]^T$ makes the expression true. This can happen as we have proved that there are functions that have correlation with those two while having no correlation to $[1,1,1,1]^T$. Lastly, we can see that correlation with an unsafe input combination is only sufficient if S is closed by $\preceq$ like all S^t.
From that expression we can see the asymptotic execution time to calculate the first $c < W/2$ coefficients, with W the same as W^g but with no multiplicity due to copy gates:

$$O\left(2^{O^g}\binom{W^g}{c} + 2^{O^g}\binom{W}{c}(2^{O^g+c}I^g + 2^{I^g}I^g)\right)$$

While to calculate all the coefficients it's:

$$O\left(2^{O^g+W^g} + 2^{2O^g+2W}I^g\right)$$

RPC_C A possible example of an RPC-like property that can be easily proven using only the correlation matrix is the following. A given gadget g is (p,ε,S)-RPC_C iff:

$$\varepsilon \geq \underbrace{\sum_{W\in\mathbb{F}_2^{W^g}} \overbrace{p^{|W|}(1-p)^{W^g-|W|}}^{\text{prob. } W \text{ is the leakage}}}_{\text{weighted average}} \text{is} \left[\underbrace{\exists \omega \in \mathbb{F}_2^{O^g} : \text{Safe}_S^g(\omega)}_{\text{Any safe out}} \underbrace{\exists \alpha \in \mathbb{F}_2^{I^g} : \neg\text{Safe}_S^g(\alpha)}_{\text{Any unsafe input}} \overbrace{\underbrace{\exists \psi \preceq W}_{\text{Any sub probe}}}^{\text{Due to selection}} \overbrace{C^g_{\omega|\psi,\,\alpha} \neq 0}^{\text{Any correlation}}\right]$$

This is quite different from the others. Both the input and the output are used directly and without checking the sub-probes, even if $\text{Safe}^g_{S^t}(\omega)$ is not a valid translation of a probability distribution for any $t > 0$. It also iterates over the safe O inside the is $[\cdots]$ and it iterates over the unsafe inputs. Those things together also imply that this property and RPC_VRAPS are neither one more accurate than the other, and their accuracy depends on the specific gadget.
From that expression we can see the asymptotic execution time to calculate the first $c < W/2$ coefficients, with W the same as W^g but with no multiplicity due to copy gates:

$$O\left(\binom{W^g}{c} + 2^{O^g}\binom{W}{c}(2^c + 2^{I^g}I^g)\right)$$

And to calculate all coefficients the complexity is:

$$O\left(2^{W^g} + 2^{O^g+W}(2^W + 2^{I^g}I^g)\right)$$

RPC_SD A RPC-like property that is more accurate than RPC_VRAPS can be obtained by using the SD, like the definition of RPS. In short, where the RPC_VRAPS requires the SD to be 0 (identical distribution), this one gives it an upper bound. Where RPC_VRAPS gives an upper bound on the probability that the input is unsafe, this one requires that it's 0.
More formally, a gadget g is (p, ε, S)-RPC_SD if exists the function in that given $o \in \mathbb{F}_2^{O^g}, w \in \mathbb{F}_2^{W^g}$ returns the safe inputs $I \in \mathbb{F}_2^{I^g} : \text{Safe}_S^g(I)$, and exists a probabilistic simulator $Sim^g(O, W, I, x \cap I) \preceq (O, W)$ such that:

$$\varepsilon \geq \underbrace{\max_{x \in \mathbb{F}_2^{I^g}}}_{\text{For all in}} \underbrace{\max_{\substack{O \in \mathbb{F}_2^{O^g} \\ :\text{Safe}_S^g(O)}}}_{\text{For all safe out}} \underbrace{\sum_{W \in \mathbb{F}_2^{W^g}} \overbrace{p^{|W|}(1-p)^{W^g-|W|}}^{\text{prob. } W \text{ is the leakage}} \overbrace{\underset{I=in(O,W)}{\text{let}}}^{\text{From the def}} \text{SD} \left[\overbrace{Sim^g(I, O, W, x \cap I)}^{\text{simulator}}; \overbrace{g(x) \cap (O, W)}^{real} \right]}_{\text{SD with fixed } O,W}}_{\text{weighted average}}$$

This is nearly the same as Equation (4), which is equivalent to the definition of RPS. The differences are:

- there is the $\max_O$ like in RPC_VRAPS as the upper bound must happen for all safe output combinations.
- The $\max_x$ is over all masked inputs like in the definition of RPC_VRAPS given in Section 2, which means the $\text{Enc}^g(x)$ is no longer used.
- The simulator receives the $x \cap in(O, W)$, this too like the RPC_VRAPS in Section 2.

Like for Expression (5) we define the useful sub-expression of RPC_SD:

$$SD_{rpc\_sd}^{g,O,W,I,x} = \underbrace{\text{SD} \left[\overbrace{Sim^g(I, O, W, x \cap I)}^{\text{simulator}}; \overbrace{g(x) \cap (O, W)}^{real} \right]}_{\text{SD with fixed } O,W,I}$$

A difference with the RPS is that for RPC_SD we always use the Sim^g obtained from the real gadget by setting the unknown bits to the uniform distribution (similar to RPS_COR1):

$$Sim^g(I, O, W, x') = g(x' \cup \underbrace{(\leftarrow \mathbb{F}_2^{I^g} \cap \neg I)}_{\text{missing bits are random}}) \cap (O, W)$$

And this leads to:

$$SD_{rpc\_sd}^{g,O,W,I,x} = \overbrace{2^{-|O|-|W|}}^{\text{from } \mathcal{F}^{-1}[\mathcal{F}[.]]} \underbrace{\frac{1}{2} \sum_{o \preceq O|W}}_{\text{the SD's}} \left| \underbrace{\sum_{\omega \preceq O|W}}_{\text{all subsetsof } O,W} \underbrace{\sum_{\alpha \in \mathbb{F}_2^{I^g} : \alpha \not\preceq I}}_{\substack{\text{any coord. that} \\ \text{selects bit outside of } I}} C_{\omega,\,\alpha}^g \overbrace{H_{o,\,\omega} H_{\alpha,\,x}}^{\text{from } \mathcal{F}^{-1}[\mathcal{F}[.]]} \right|$$

While this may seem much more complex, it's just like RPC_VRAPS with the same difference found between RPS_COR3 and RPS_COR1. Just like the latter, it considers both the value and the position of each component of the correlation matrix, making it more accurate.
We can also use this simulator to prove that for RPC_SD there is an upper bound similar to Expression (6):

$$SD_{rpc\_sd}^{g,O,W,I,x} \leq 1 - 2^{-(I^g - |I|)} \tag{9}$$

RPC_SD_COR1 According to RPC_SD, to obtain the minimal ε it would be necessary to minimize that expression over the function in. A possible approximation to avoid that is:

$$\varepsilon \geq \underbrace{\max_{\substack{O \in \mathbb{F}_2^{O^g} \\ :\text{Safe}_S^g(O)}}}_{\text{For all safe out}} \underbrace{\sum_{W \in \mathbb{F}_2^{W^g}} \overbrace{p^{|W|}(1-p)^{W^g-|W|}}^{\text{prob. } W \text{ is the leakage}}}_{\text{weighted average}} \underbrace{\min_{I \in \mathbb{F}_2^{I^g} : \text{Safe}_S^g(I)}}_{\text{Bringing in the } \exists} \underbrace{\max_{x \in \mathbb{F}_2^{I^g}}}_{\text{For all in}} \underbrace{\text{SD} \left[\overbrace{Sim^g(I, O, W, x \cap I)}^{\text{simulator}}; \overbrace{g(x) \cap (O, W)}^{real} \right]}_{\text{SD with fixed } O,W}$$

Where the only approximation happens because bringing the $\max_x$ inside a sum raises the result. During this operation we left the expression of $SD_{rpc\_sd}^{g,O,W,I,x}$ unaltered from RPC_SD.
From this expression we can see the asymptotic execution time to calculate the first $c < W/2$ coefficients, with W the same as W^g but with no multiplicity due to copy gates:

$$O\left(2^{O^g} \binom{W^g}{c} + 2^{O^g} \binom{W}{c} (2^{2O^g + 2c + 2I^g} + 2^{3I^g})\right)$$

While to calculate all coefficients is (assuming all inputs are linked to a gate):

$$O\left(2^{O^g + W^g} + 2^{3O^g + 3W + 2I^g}\right)$$

RPC_SD_COR2 From RPC_SD_COR1 We can derive a similar approximation to RPC_VRAPS (with $t = \max_{s \in S} |s|$):

$$\varepsilon \geq \overbrace{(1 - 2^{-I^{U^g}(d^g - t)})}^{\text{upper bound}} \overbrace{\max_{\substack{O \in \mathbb{F}_2^{O^g} \\ : \text{Safe}_S^g(O)}}}^{\text{All safe out}} \overbrace{\sum_{W \in \mathbb{F}_2^{W^g}} \overbrace{p^{|W|}(1-p)^{W^g - |W|}}^{\text{prob. } W \text{ is the leakage}}}^{\text{weighted average}} \overbrace{\text{is } \left[\overbrace{\forall I \in \mathbb{F}_2^{I^g} : \text{Safe}_S^g(I)}^{\text{All possible results of } in}, \underbrace{\exists \alpha \in \mathbb{F}_2^{I^g} : \alpha \not\preceq I}_{\substack{\text{any coord. that} \\ \text{selects bit outside} \\ \text{of } I}} \overbrace{\exists \omega \preceq (O|W)}^{\text{Due to the out selection}} \overbrace{C^g_{\omega,\,\alpha} \neq 0}^{\text{Any correlation}} \right]}^{\text{All possible result of } in \text{ have a dependency they don't cover}}}^{\text{same as RPC\_VRAPS}}$$

Where $\max_x$ is no longer needed due to the approximations making x disappear, and this allows $\min_I$ to be moved in the is []. As per the upper bound, it's slightly different from Expression (9) as it uses the lowest possible upper bound (due to the $\min_I$).
This is very close to the expression of RPC_VRAPS except for a coefficient that makes it lower. Yet while RPC_VRAPS is RPC-like, I did not prove that RPC_SD_COR2 is RPC-like, but only that it's an approximation of RPC_SD, which is RPC-like. For this reason, they're different things, and RPC_SD_COR2 can't be seen as a direct improvement of RPC_VRAPS, even if more accurate.
Lastly, we can see that RPC_SD_COR2 has the same asymptotic execution time as RPC_VRAPS.

RPC_C_L The last approximation is obtained from RPC_C using the L^g and with the objective to speed up the calculation of the whole function.

$$\varepsilon \geq \underbrace{\sum_{i \in \mathbb{Z}_{W^g+1}}}_{\substack{\text{over possible} \\ \text{coefficients}}} \overbrace{p^i(1-p)^{W^g - i}}^{\substack{\text{prob. of one} \\ \text{combination with} \\ i \text{ wires is leaking}}} min \underbrace{\left(\overbrace{\binom{W^g}{i}}^{\substack{\text{combinations} \\ \text{with } i \text{ wires}}}, \underbrace{\sum_{\omega \in \mathbb{F}_2^{O^g} : \text{Safe}_S^g(\omega)}}_{\text{Any safe out}} \underbrace{\sum_{\alpha \in \mathbb{F}_2^{I^g} : \neg\text{Safe}_S^g(\alpha)}}_{\text{Any unsafe input}} \overbrace{(L^g_{\omega\ \alpha})_i}^{\text{compact correlation matrix}} \right)}_{\text{coefficient } c_i \text{ of } \varepsilon(p)}$$

This is similar to RPS_L with two differences. The first is that there is no upper bound that multiplies the rest to decrease the expression, and this is because the upper bound comes from the SD, which is not used in RPC_C. The second is which columns and rows it selects.
Like for RPS_L this is supposed to be calculated by composing the L^g, and its accuracy will be lower than COR_C. Yet the asymptotic complexity to calculate all coefficients for a circuit of width w and with l non-copy gates is:

$$O\left(2^w l W^{g\,2}\right)$$

and is exactly the same as that of RPS_L.
As $l = W - I^g - R^g + O^g$ and $w \leq l + I^g + R^g = W + O^g$ then even with the worst possible width w

$$O\left(2^{W+O^g}(W + O^g)W^{g\,2}\right)$$

An assumption that covers nearly all gadgets is that the number of non-copy gates is at least that of the number of inputs. If that happens, then $W > O^g$ This means

$$O\left(2^{O^g+W} W W^{g\,2}\right)$$

Which is quite lower than the other best complexity, i.e. that of RPC_C:

$$O\left(2^{W^g} + 2^{O^g+W}(2^W + 2^{I^g} I^g)\right)$$

The following images show the relationship between the various approximations in terms of accuracy and execution time. As established at the start of this section, the accuracy between different definitions is in terms of the accuracy of the implied RPS.

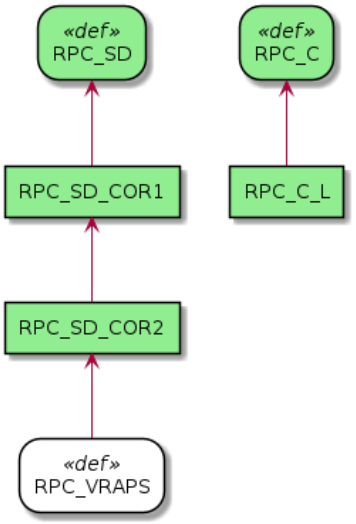


Figure 3: Accuracy, arrows toward the higher accuracy. Light green for the newly introduced definitions/approximations.

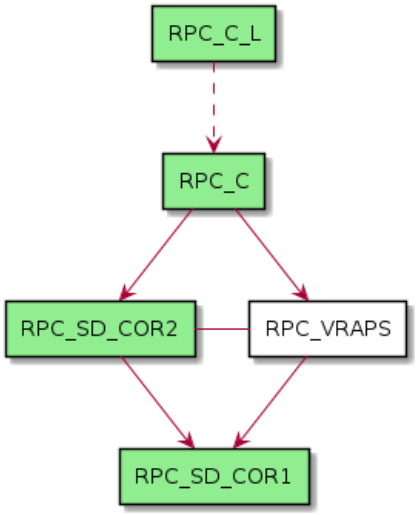


Figure 4: Asymptotic timing: arrows toward the higher complexity. Links if the complexity (as calculated here) is the same. If dashed, it only refers to the complexity to calculate the whole function, while the others also include when calculating fewer coefficients. Light green for the newly introduced definitions/approximations.

4. Analyzing the performance and accuracy of the new models

This section analyzes the execution time and accuracy trade-off that we described theoretically in Section 3. To this end, we show the graphs of various of the gadgets present in the literature of the random probing security. We also use those graphs to compare our tool with the known alternatives.

4.1. Overview

This subsection explains the various aspects of the graphs: which tools are available and their basic characteristics, why we didn't use all of them, how we reduced the accuracy (a function) to a single scalar value, and which gadgets we used.

4.1.1 Tools

RPSC We have created a software tool to compare the various expressions presented in this thesis. It's based on the correlation transform, and it has at most the asymptotic complexities of Section 3. It can be found at https://github.com/ManzoniGiuseppe/random_probing_security_checker/tree/thesis.

VRAPS We compare the results of our tool (both time and accuracy) with those produced by VRAPS. This was the first tool that provided random probing-like properties, and it was introduced with [5]. It supports any gadget, but it may return coefficients higher than the minimum, as it can have false positives when analyzing the dependencies of a set of wires [6] [9].

STRAPS Straps is a more recent tool that calculates PDT and uses them to calculate the RPS of a gadget or a circuit. It calculates part of the results probabilistically using Monte-Carlo methods, which allow to improve the performance by avoiding a full exploration, and these methods also assure that the result is correct with a given probability of error (passed as a parameter). Like VRAPS it can return an f(p) higher than the minimal one. We have not used this tool as, to the best of our knowledge, it does not provide any numeric result: neither the coefficients of the function, any $(p, f(p))$ pairs, nor the $\max_p : f(p) \leq p$. Instead, it only returns a graph plotting the f(p), making this tool unsuitable to analyze the accuracy and execution time trade-off.

IronMask The lack of IronMask's results is intentional, and the most pressing reason is that it doesn't seem to work: it usually returns a set of coefficients lower than what we have proved to be the theoretical minimum. While we may have made a mistake in our proof or in our tool, the following example with a minimal gadget (2-shares addition) should show that our eventual mistake doesn't change that IronMask returns an f(p) lower than the definition's minimum. To do this, we calculate by hand the RPC with $t = 1$, and in particular we only calculate a lower bound (considering only $J_0 = \{d0\}$) using the definition. IronMask's coefficient for $p^0(1-p)^9, p^1(1-p)^8$ are correctly 0, but the coefficient returned for p^2 is 7. Follow the '.sage' code and the dependencies of the single probes and outputs:

```
#SHARES 2
#IN a b
#RANDOMS r0
#OUT d

c0 = a0 + b0
c1 = a1 + b1

d0 = c0 + r0
d1 = c1 + r0
```

wire	a0	a1	b0	b1	r0	wire multiplicity
a0	x					1
a1		x				1
b0			x			1
b1				x		1
r0					x	3
c0	x		x			1
c1		x		x		1
d0	x		x		x	
d1		x		x	x	

For each wire and output, its multiplicity and which input and random it depends on.

Where r0 has 3 copies due to the need for a copy gadget (as it's used twice). This is correct, as IronMask itself agrees on the total number of wires and the dependencies shown here.
According to the RPC definition (RPC_VRAPS), a failure happens whenever both shares on either input are needed to simulate the output and the leaking wires, and the coefficient of $p^2(1-p)^7$ is equal to the number of combinations with two leaking wires.
We can see for the output $d0$ that the combinations reported in the following table are the failing ones, in particular for each combination we provide a f that calculates a value using the wires visible to the attacker, and then we report the dependencies of the output of that f.

multiplicity	out	W0	W1	f(out,W0,W1)	dependencies of f(leakage)
1	d0	a0	a1	W0 + W1	**a0 + a1**
1	d0	a0	c1	W0 + W1	**a0 + a1** + b1
1	d0	a1	c0	W0 + W1	**a0 + a1** + b0
1	d0	b0	b1	W0 + W1	**b0 + b1**
1	d0	b0	c1	W0 + W1	a1 + **b0 + b1**
1	d0	b1	c0	W0 + W1	a0 + **b0 + b1**
1	d0	c0	c1	W0 + W1	**a0 + a1 + b0 + b1**
3	d0	r0	a1	out + W0 + W1	**a0 + a1** + b0
3	d0	r0	b1	out + W0 + W1	a0 + **b0 + b1**
3	d0	r0	c1	out + W0 + W1	**a0 + a1 + b0 + b1**

It stands to reason that if the leakage depends on a given set of inputs, applying a $f : \mathbb{F}_2^3 \to \mathbb{F}_2^1$ to it can't add new dependencies, as that function doesn't take anything as input except the leakage. This means in addition to the first 7 combinations, don't depend on any random, the following 9 (3*3) must be considered too, as the two appearances of the same random disappears when considering the xor of all wires and output.
This means the right coefficient for $p^2(1-p)^7$ is ≥ 16 (it's actually =16) and not 7 as returned by IronMask.

4.1.2 Accuracy

In Section 3, the accuracy of an expression (an approximation of RPS or of an RPC-like property) is obtained by comparing, for all gadgets g, the minimal f of that expression, and this comparison is meaningful because that's the best f that each expression can guarantee for the RPS of g.
As we can see in all expressions, the $f(p)$ can be written as

$$f(p) = \sum_k p^k (1-p)^{W^g - k} c_k \tag{10}$$

Where each c_k is upper bounded by $\binom{W^g}{k}$ (as explained in [5]) times the upper bound on the SD (wherever present). As all the properties check that f is greater or equal to something, we can approximate the $f(p)$ by calculating the first coefficients while fixing the rest to the upper bound.

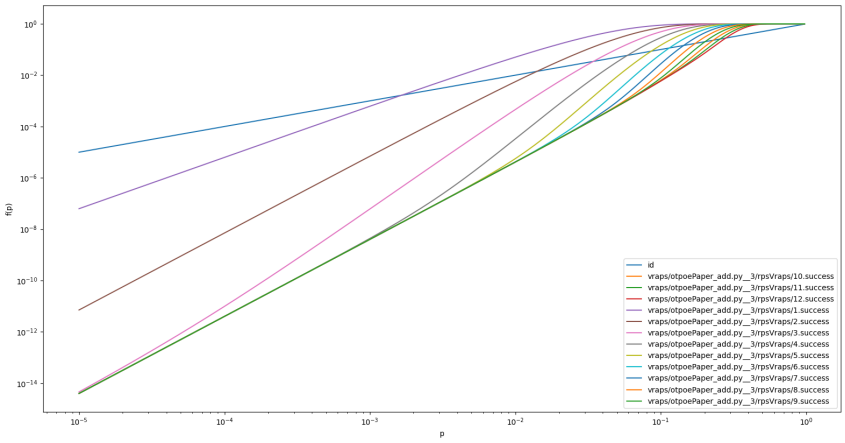


Figure 5: $f(p)$ varying the coefficients. With the identity function f(p)=p.

The Figure 5 shows what happens when calculating a different number of coefficients. The straight line that intersects all the others is the identity function, to point out where $f(p) = p$. As per the other functions, if we order them from the left to the right in the same order they intersect the identity function, then the maximum coefficient they calculate grows gradually from c_1 to c_{12}.
We can see that at first it's the slope for $p \to 0$ that varies, and this stops at the first coefficient $\neq 0$. Then all the following functions have the same asymptotic behavior, and the more coefficients are added, the more the area affected by the new coefficient moves a bit more to the right.
To plot the accuracy and execution time it's necessary to reduce a calculated function $f : [0, 1] \to [0, 1]$ into a single value, and we chose to use the maximum tolerated leakage:

$$\max_{p \in (0,1) \wedge f(p) \leq p} p$$

This is significant for RPS and RPC because for all higher p, compiling the circuit with this gadget worsens the security, while with lower p the gadget improves the security of the circuit. This can be seen in Figure 5 as the point where a given $f(p)$ crosses the *id* function (excluding p=1). That figure also shows how the maximum tolerated leakage is quite sensible to the precision of the function, and a lower function has a higher tolerated leakage and a higher accuracy.
It's also possible to calculate a limit to the improvement in accuracy obtained by calculating all the remaining coefficients. This can be done by setting all the non-calculated coefficients to 0 instead of the maximum, as no function can go lower.

4.1.3 Gadgets

We tested the tool on various gadgets. In particular, while [5] describes two addition gadgets, the formulas don't match the descriptions due to the lack of parenthesis, so we included all 4. They create plots with similar shapes, so we only show one of them. We also added the circular refresh that those gadgets internally use. In contrast to this, there is the circular refresh from [7], which suggest adding the randoms together first, instead of adding them to the shares one at a time. For the RPC, we used $S^t, t = \frac{d^g-1}{2}$ as the middle way (and the optimal one for RPE).

paper	gadget	d^g	W^g	W^g without copies	R^g
[11]	multiplication	3	57	27	3
		5	180	80	10
[5]	multiplication	3	97	51	11
	copy	3	33	15	6
	add1	3	36	24	6
	add2	3	36	24	6
	add3	3	36	24	6
	add4	3	36	24	6
	circular refresh	3	15	9	3
[7]	small multiplication	3	127	69	17
	multiplication	3	132	72	18
		5	380	200	50
	small copy	3	23	9	4
	copy	3	33	15	6
		5	55	25	10
	small add	3	26	18	4
	add	3	36	24	6
		5	60	40	10
	small refresh	3	10	6	2
	circular refresh	3	15	9	3
		5	25	15	5
	isw refresh	3	15	9	3
		5	50	30	10

4.2. Single-thread RPS

Beside the actual results of VRAPS, we compare RPS_COR1, RPS_COR2 and RPS_COR3. We don't show the comparison with RPS_VRAPS because we can obtain it with the accuracy of VRAPS (visually the same) and the time of RPS_COR3 (which has a higher accuracy). We believe adding it would just make the graphs less readable. We don't show RPS_IRONMASK either, as its accuracy is worse than that of RPS_VRAPS and it's slower, so is not relevant for the trade-off. RPS_L is missing because we didn't implement it, as explained at the start of 3.2.

4.2.1 Multiplication

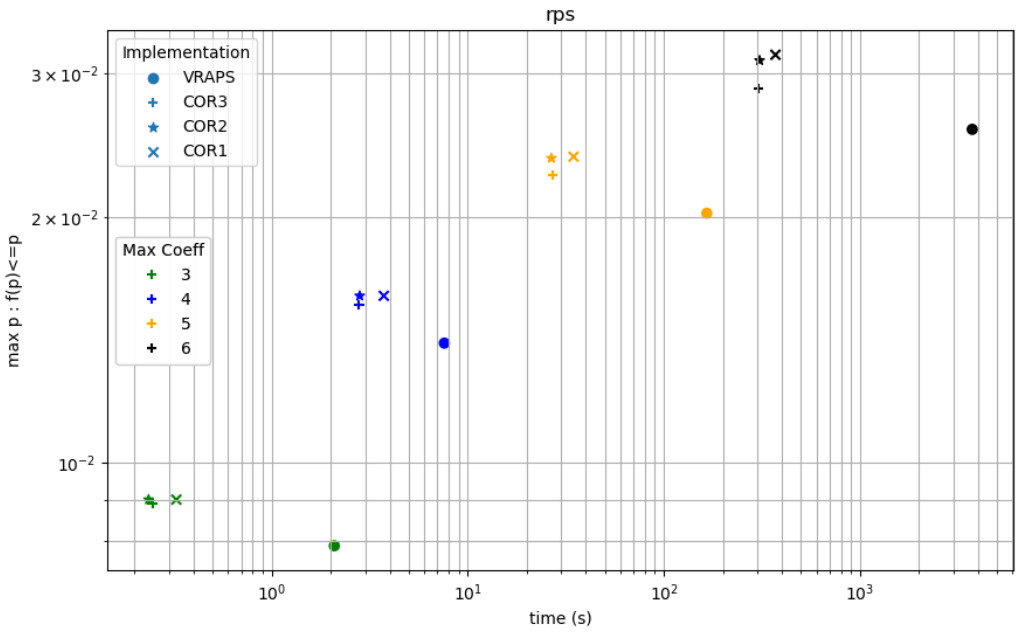


Figure 6: Plot in logarithmic scales of execution time and accuracy of [5]'s multiplication, varying the number of coefficients.

In Figure 6 we can see [5]'s multiplication gadget, and in this case even the RPS_COR1 is asymptotically faster than VRAPS. We can see how in this case the VRAPS tool gets progressively slower than the other three functions computed by our tool, if one excludes the c_3 as VRAPS has a higher startup time.

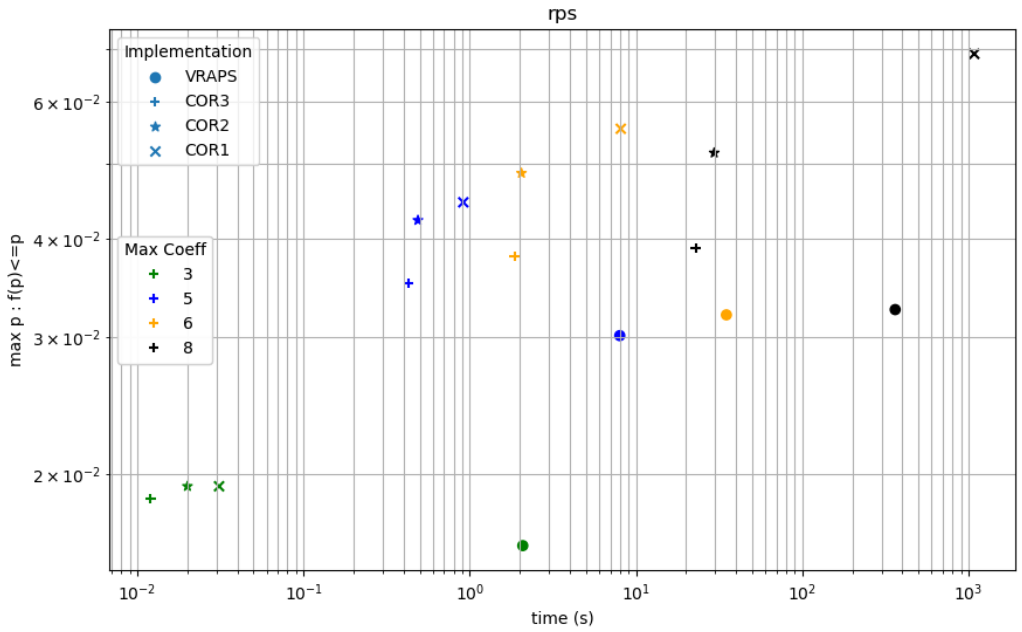


Figure 7: Plot in logarithmic scales of execution time and accuracy of ISW's multiplication [11] with 3 shares, varying the number of coefficients.

The Figure 7 shows in the bottom (in green) the four functions calculated up to coefficient c_3. The others are obtained by increasing the number of coefficients calculated, and we can see how at each step they increase in accuracy and computation time. At c_8 we obtain the final accuracy (except for RPS_COR1, which could still improve). We can also see how increasing the coefficient makes RPS_COR1 become progressively slower than the VRAPS tool, while the others remain a steady order of magnitude faster.

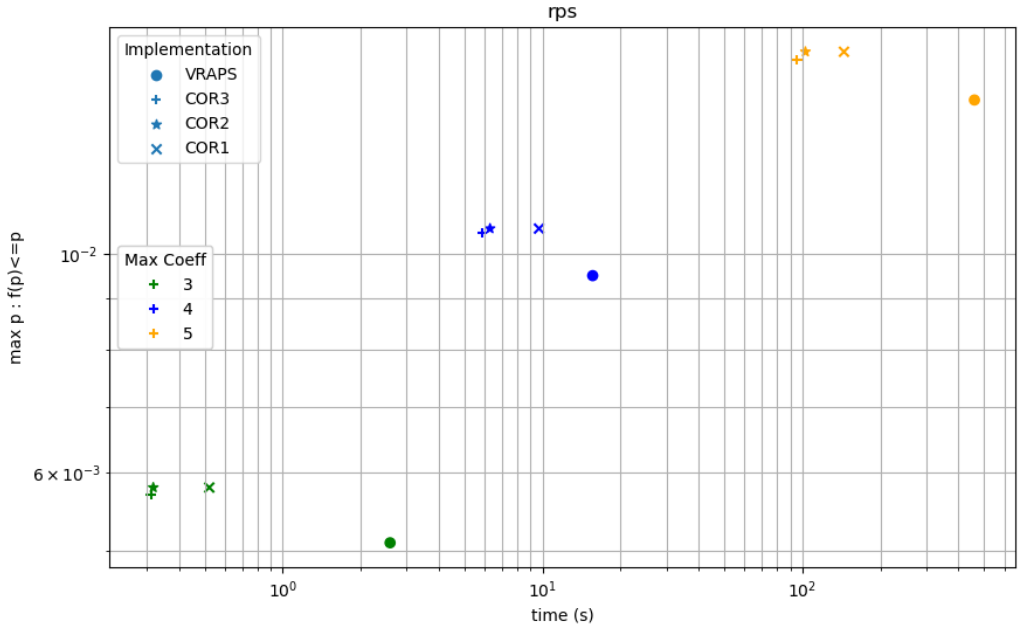


Figure 8: Plot in logarithmic scales of execution time and accuracy of [7]'s multiplication with 3 shares varying the number of coefficients.

The Figure 8 represent the results for [7]'s multiplication. The higher coefficients couldn't be calculated due to the program ending the memory of our computer just by storing all the hashes of the spectral indexes of

the various rows. This could be fixed by increasing the computation and slowing the program, which we didn't choose as a trade-off. VRAPS instead does calculate the function, but it takes hours to complete and without the other points it's not very informative. This is similar for the gadgets with an elevated number of probes, like the other two multiplications of [7].

4.2.2 Addition

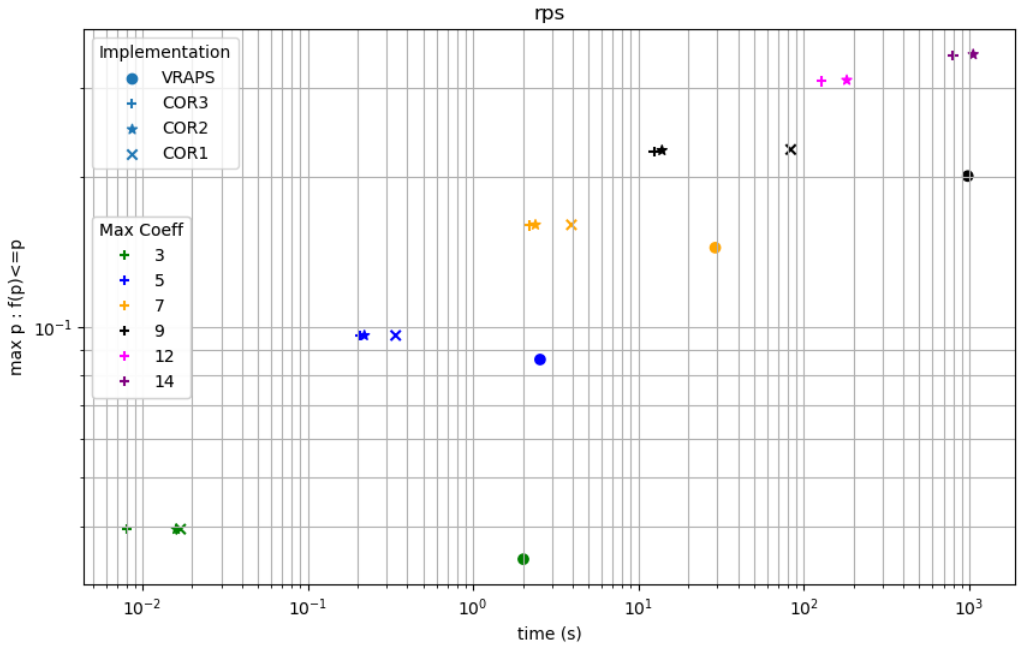


Figure 9: Plot in logarithmic scales of execution time and accuracy of one of [5]'s additions, varying the number of coefficients.

We can see in Figure 9 various aspects. Accuracy-wise, it's clear how in this gadget the differences between the four approximations is low, and only due to the upper bound on the SD. Time-wise, RPS_COR1 is initially faster than VRAPS, but it gets progressively slower, while RPS_COR2 and RPS_COR3 remain an order of magnitude faster. We also added partial results for higher coefficients to show how the faster function translates to higher accuracy when executed for a similar amount of time (by calculating more coefficients). We weren't able to plot VRAPS up to c_{10} because of memory limits.

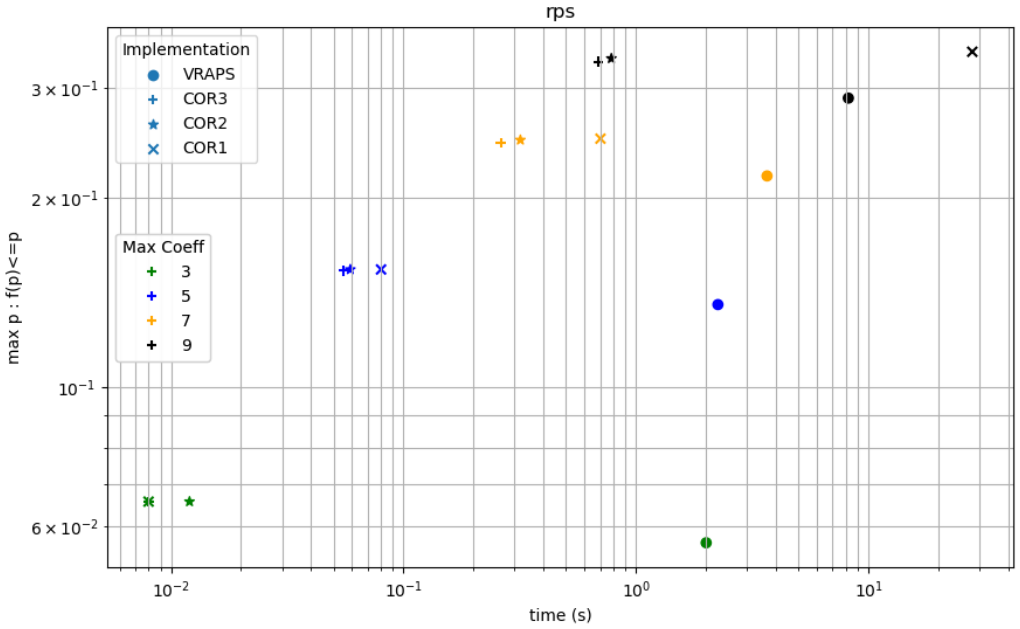


Figure 10: Plot in logarithmic scales of execution time and accuracy of [7]'s optimized addition, varying the number of coefficients.

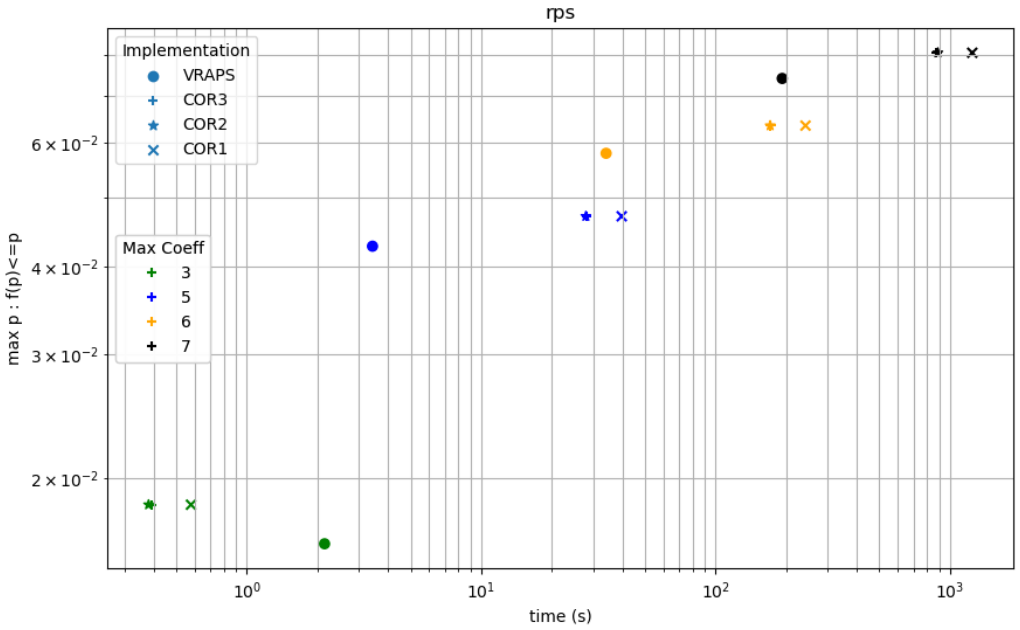


Figure 11: Plot in logarithmic scales of execution time and accuracy of [7]'s addition with 5 shares, varying the number of coefficients.

The Figure 10 shows the same patterns of RPS_COR1 being asymptotically slower than VRAPS, and of the other two being an order of magnitude faster, excluding the initial startup time of VRAPS. The same patterns are visible when using the [7]'s addition for d=3. A difference can be seen for 5 shares in Figure 11: all three functions calculated by our tool are slower than VRAPS, but the difference seems limited and not asymptotically slower, as it was clear in the previous graphs. We couldn't calculate the RPS_COR* for c_8 because of memory limits.

4.2.3 Copy

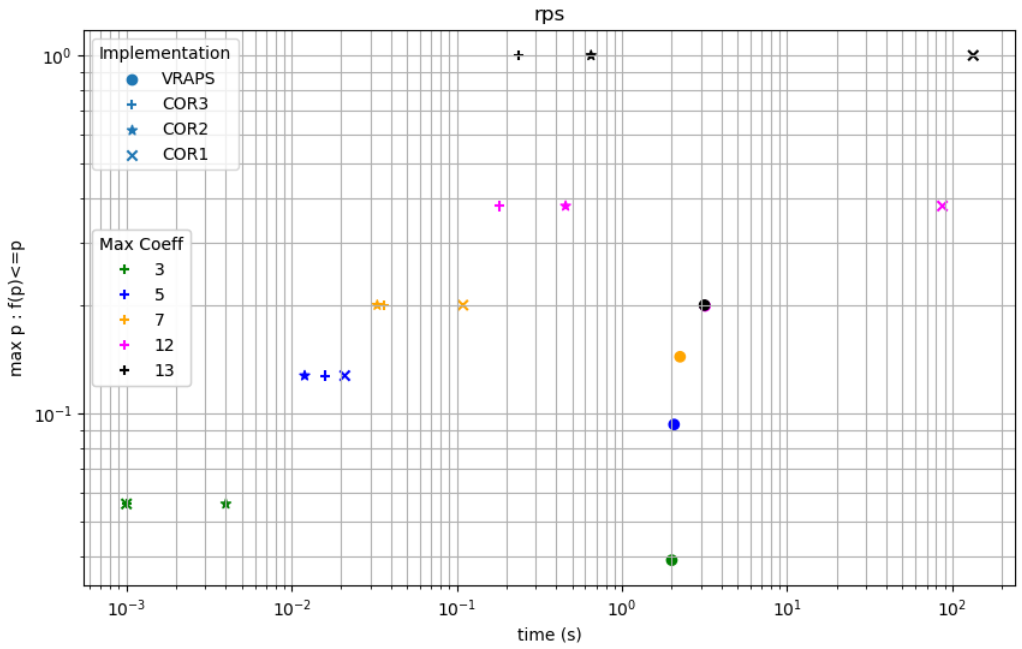
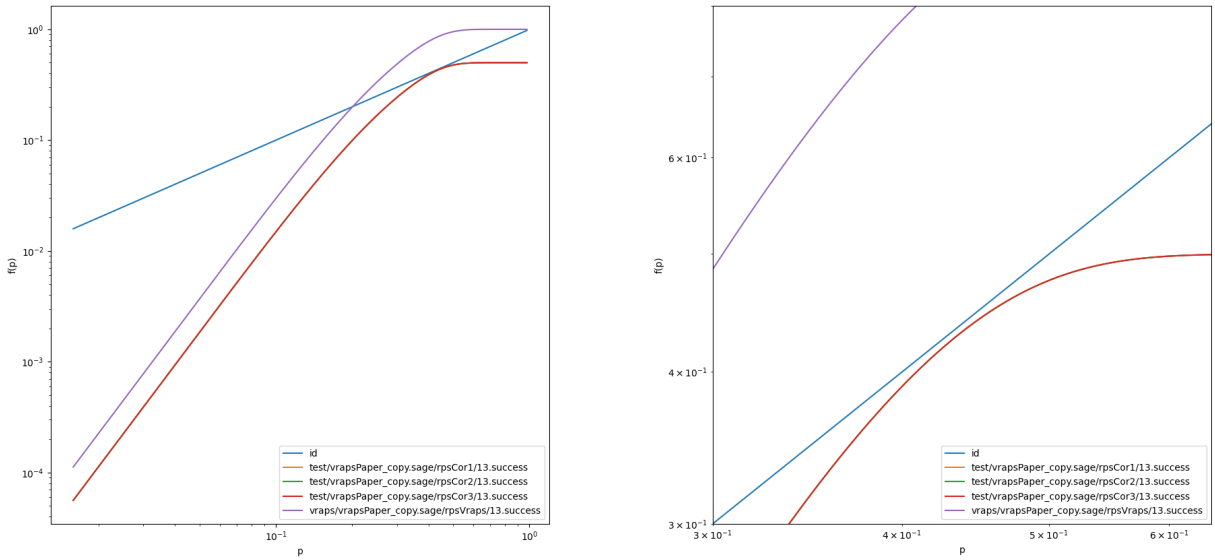


Figure 12: Plot in logarithmic scales of execution time and accuracy of [5]'s copy, varying the number of coefficients.

Figure 12 has a few things that may seem odd; VRAPS' result for c_{12} and c_{13} overlap as it reached its peak accuracy (within a margin of 0.3%). At the same time, all three of our functions reached 1.0, indicating that $\forall p, f(p) < p$. This may seem wrong, but it's due to the gadget being very small and to the Expression (6), as the upper bound on SD is 0.5.



(a) (b)

Figure 13: [5]'s copy, plot of the 4 functions up to c_{13}.

This can be seen in Figure 13 where the three functions calculated by our tool overlap and come very close to touching the $f(p) = p$ line, but remain fully below. [7]'s copy gadget optimized for d=3 is the same, just with a

bigger margin and already reaching 1.0 for c_9 (10th out of 24). The generic copy gadget with 3 shares reaches it with c_{12} (13th out of 34). Similar is the version with 5 shares, which reaches it at c_{21} (22nd out of 56).

4.2.4 Refresh

Our tool reaches the 1.0 at c_3 in all functions, in the order of magnitudes of milliseconds, while VRAPS reaches it at c_9 after 2s (also due to its high start up time). This is similar for the circular refresh, both with 3 and 5 shares.
On the other hand, VRAPS has a lower accuracy for the [7]'s ISW-based refresh. For example, with 3 shares, it reaches at most 0.63. Similar is the refresh from [5], as it reaches 0.645.

4.3. Single-thread RPC

Beside the actual results of VRAPS, we compare RPC_SD_COR1, and RPC_SD_COR2. We do not show the comparison with RPC_VRAPS because it can be obtained using the accuracy of VRAPS (visually the same) and the execution time of RPC_SD_COR2 (which has a higher accuracy). We believe adding it would just make the graphs less readable. We won't show RPC_C either, as its accuracy is nearly everywhere worse than that of RPC_VRAPS, and for the gadget tested there was no time advantage. Like for the RPS, we won't compare the RPC_SD_L as it wasn't implemented.

4.3.1 Multiplication

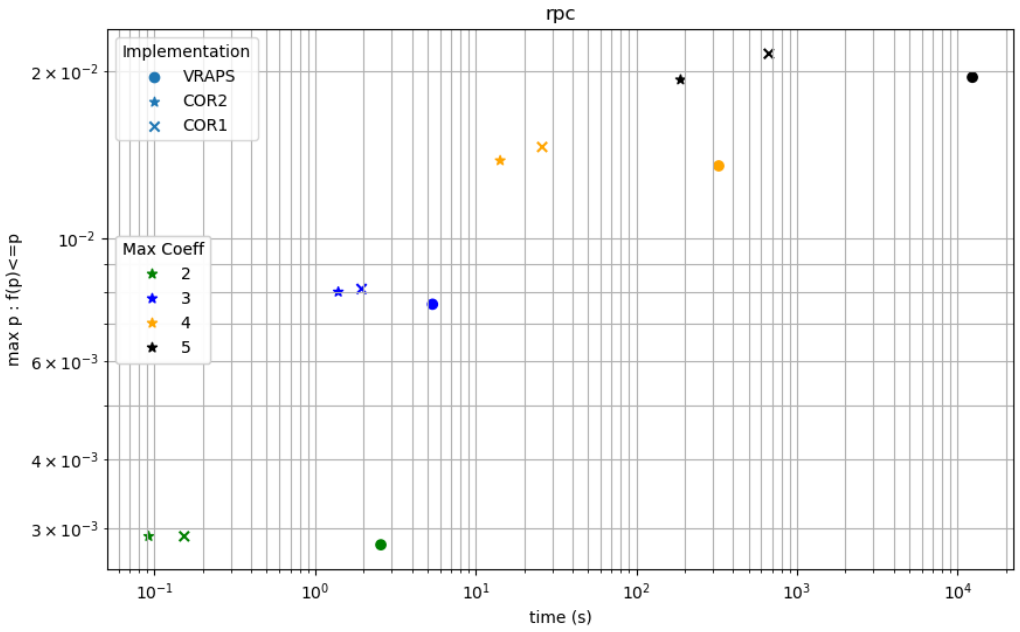


Figure 14: Plot in logarithmic scales of execution time and accuracy of [5]'s multiplication, varying the number of coefficients.

In Figure 14 we can see that while the accuracy improvement isn't that impressive, the RPC_SD_COR2 is already two orders of magnitude faster than VRAPS for coefficient c_5 (6th out of 98).

We didn't plot them due to the lack of points, but (thanks to the lower bounds on f) we found that the ISW multiplication with 3 shares finds the $\max p : f(p) < p$ at c_3. This leads to 0.0024 for VRAPS, 0.0026 for RPC_SD_COR2 and 0.0048 for RPC_SD_COR1. With 5 shares our tool reached our memory limit at (c_2 and c_4), while VRAPS goes over 30 minutes with the latter coefficient (c_4).

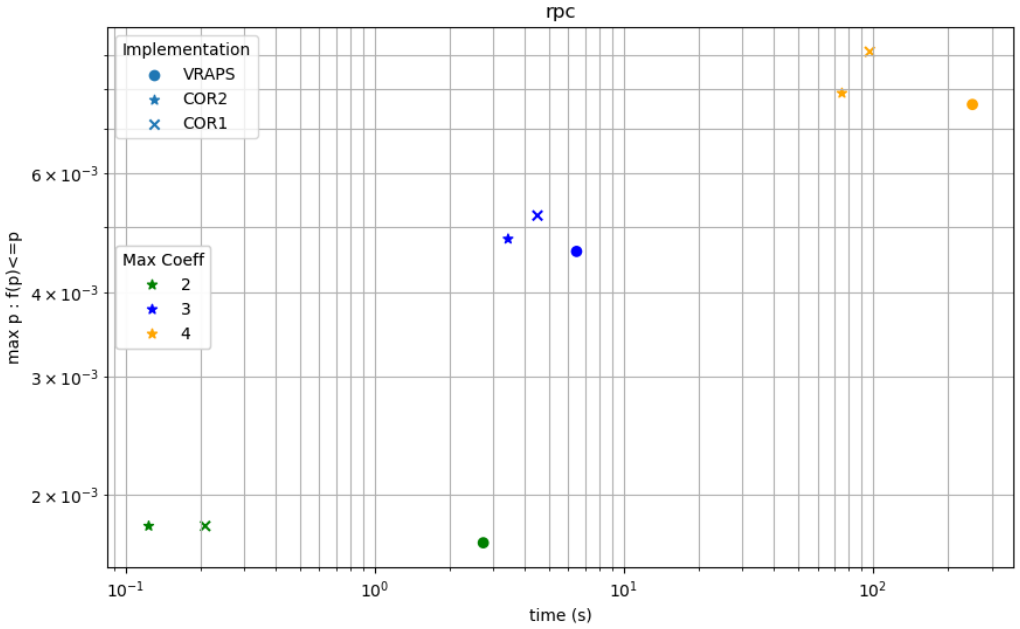


Figure 15: Plot in logarithmic scales of execution time and accuracy of [7]'s optimized multiplication, varying the number of coefficients.

In Figure 15 we can see the results for the 3 shares optimized multiplication of [7]. From the few data points we could calculate (for the higher ones we run out of memory), it seems that VRAPS is getting progressively slower than RPC_SD_COR*, and from the decrease in the rise in accuracy we know the points are getting closer to maximum. Lastly, we can estimate that VRAPS could take 2h45min to calculate c_5, while for the other two could take around 50min, if we had more memory and the trend continues.

4.3.2 Addition

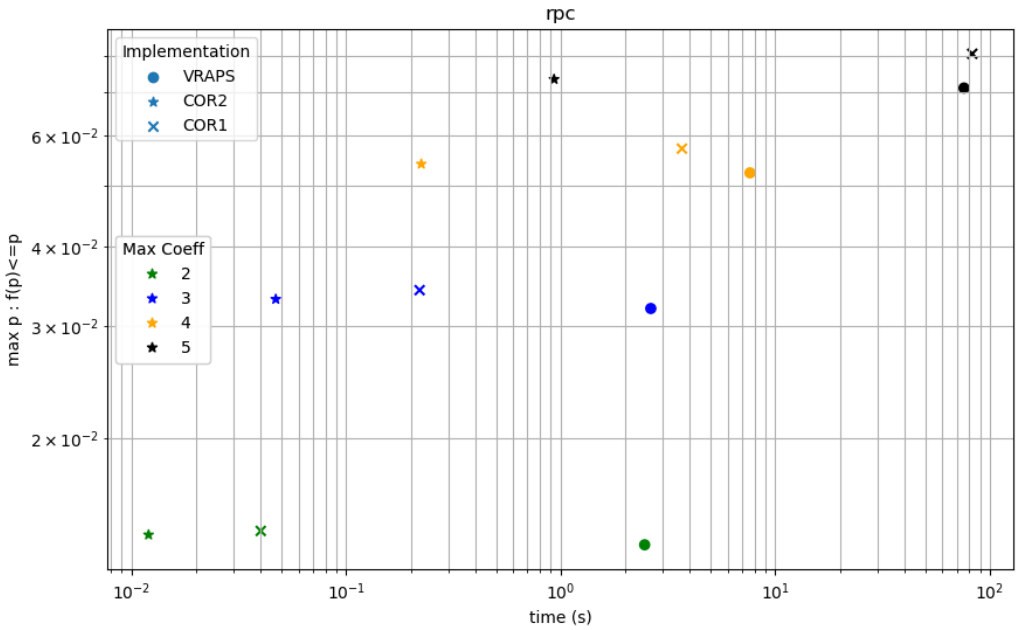


Figure 16: Plot in logarithmic scales of execution time and accuracy of one of [5]'s additions, varying the number of coefficients.

While Figure 16 was made with one of the additions of [5], all of them have graphs that look alike: with the differences slight in the accuracy, but wide in the execution time.

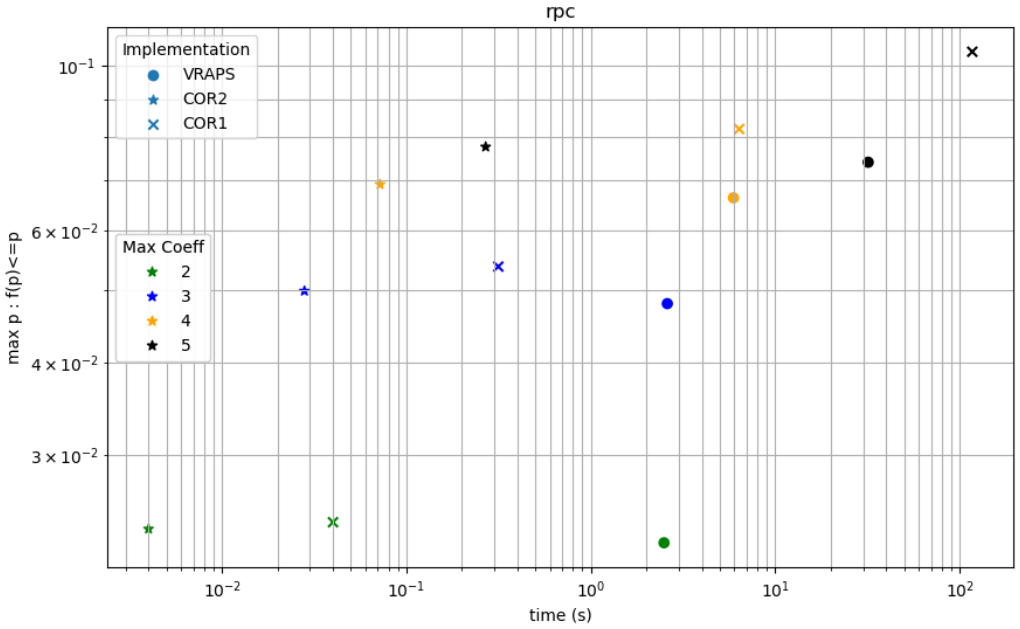


Figure 17: Plot in logarithmic scales of execution time and accuracy of [7]'s optimized addition, varying the number of coefficients.

In Figure 17 we can see the results for the 3 shares optimized addition, and how in this case RPC_SD_COR1 has a more significant improvement in accuracy than the one in Figure 16. As per the other of [7]'s additions with 3 shares, the graph looks like Figure 16.

4.3.3 Copy

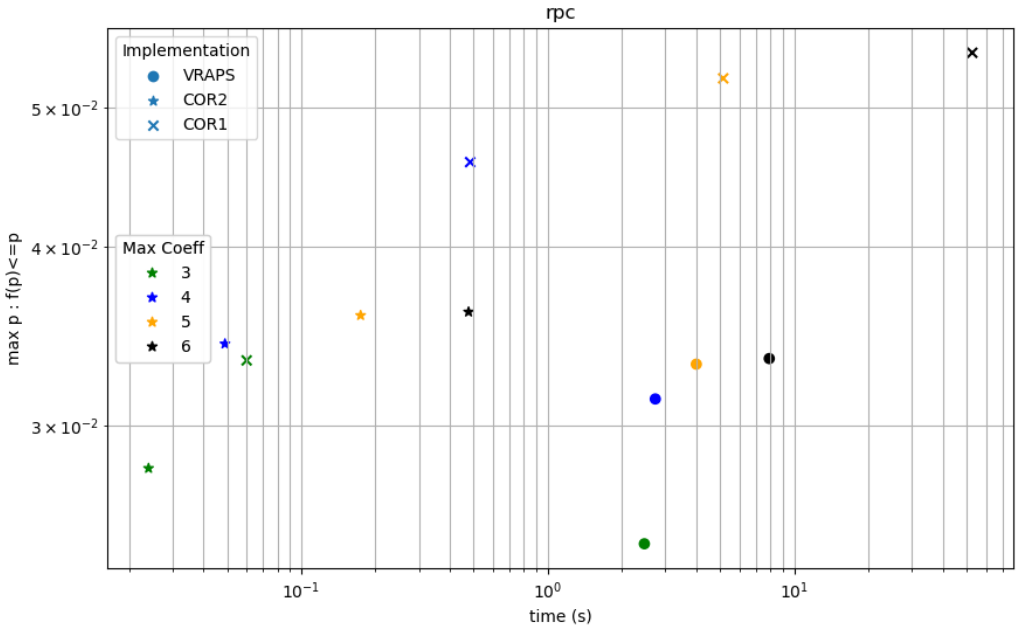


Figure 18: Plot in logarithmic scales of execution time and accuracy of [7]'s copy with 3 shares, varying the number of number of coefficients.

We can see in Figure 18 By confronting the yellow with the black points that it nearly reached its maximum accuracy (confirmed by the lower bound on the f), and that while RPC_SD_COR1 is quite slower than VRAPS, it also provides a quite higher accuracy.

4.3.4 Refresh

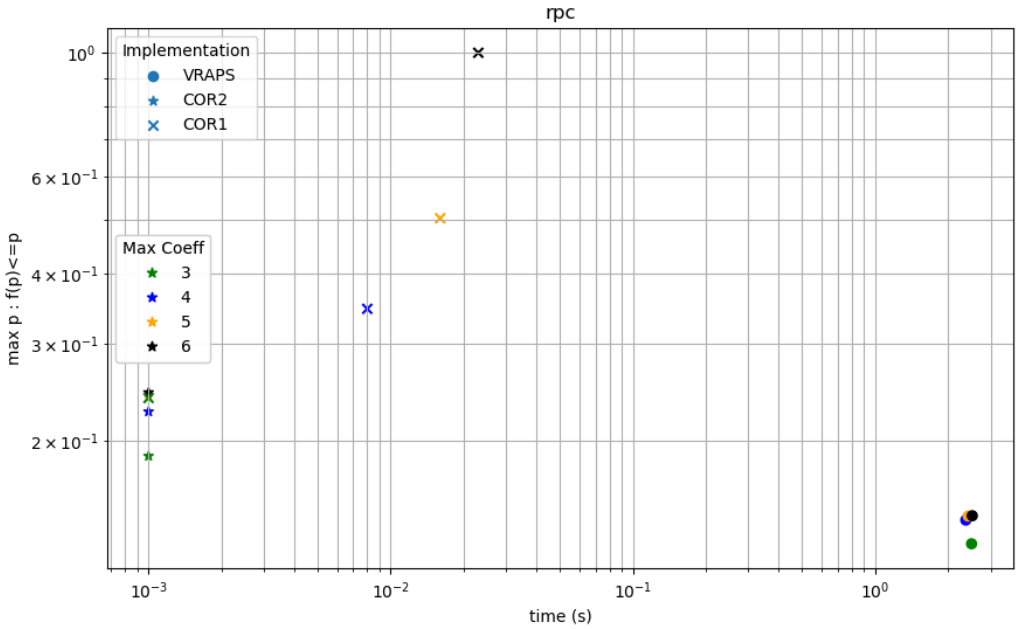


Figure 19: Plot in logarithmic scales of execution time and accuracy of [7]'s optimized refresh, varying the number of coefficients.

The optimized refresh for 3 shares can be seen in Figure 19. It's worth notice that the RPC_SD_COR2's execution time was measured as $0.000s$ and have been plotted as $0.001s$ as it's within the measurement error. It's also worth to notice that thanks to the RPC_SD_COR1 we know that $\forall p, f(p) < p$.
The circular refresh with 3 shares is similar to Figure 19, just with the RPC_SD_COR1 being less than an order of magnitude slower than VRAPS.

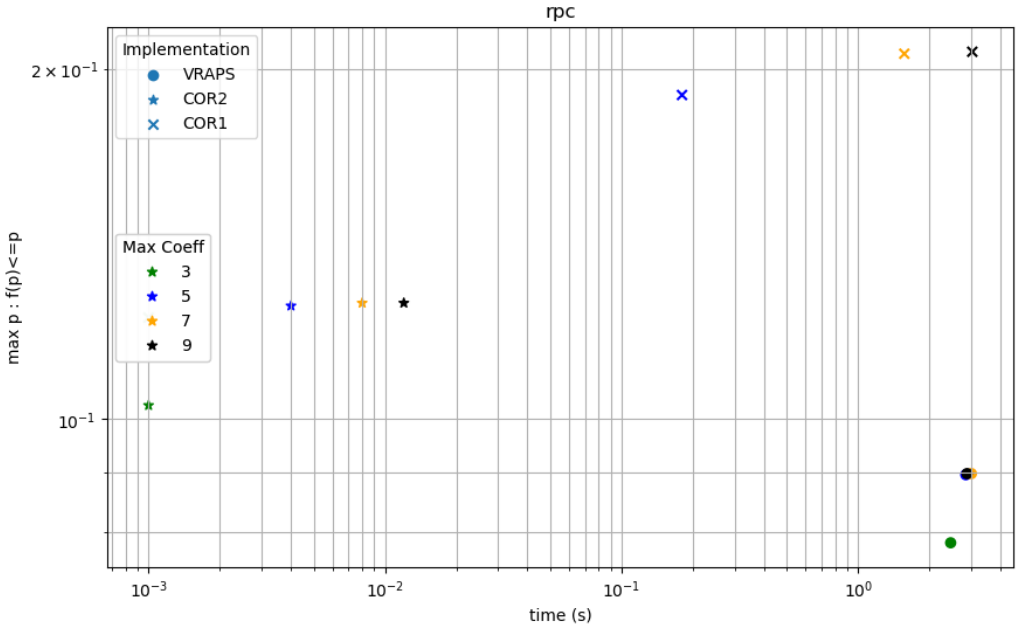


Figure 20: Plot in logarithmic scales of execution time and accuracy of [7]'s ISW based refresh with 3 shares, varying the number of coefficients.

In Figure 20 we can see that both RPC_SD_COR2 and VRAPS quickly reach their maximum accuracy, and the latter is half that of RPC_SD_COR1.
This gadget has a plot with similar relative position as Figure 20.

4.4. Multithread

While not central to the thesis, we have parallelized various parts of the tool to speed up the execution time. In particular, the tool alternates serial parts to parallel parts with T threads, all that do the same things.
We have tested the parallel execution for 2,3, and 4 threads (our machine has 4 cores) using the highest coefficients calculated for the graph of the previous sections. We have also ignored tests with a serial execution of less than a second to decrease the measurement errors.
To measure this the speedup, we have used the 'time' command. This returns three values. 'real' indicates the difference between the program start and the program end. In addition, we can sum the other two ('sys' and 'user', and call it 'execution') to obtain the total time the CPU spent executing the program (the sum of all the threads), both for serial and parallel implementations.
As the 'real' time indicates the difference between the start and end, it includes time spent executing the background processes. For the considered serial programs (where it can be easily measured), the maximum time increase of 'real' over 'execution' is 0.065%, so I'll be ignoring it.

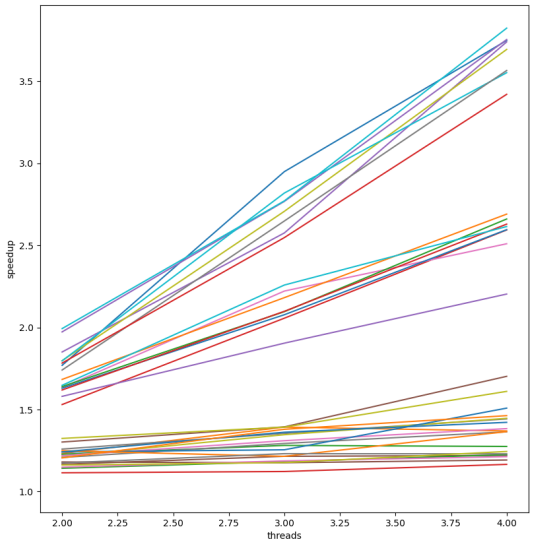


Figure 21: Speedup for a given the number of threads.

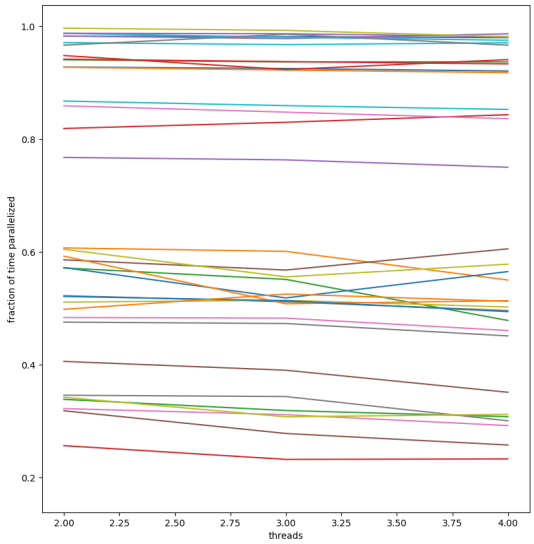


Figure 22: For a given number of thread, the estimate of the fraction of the serial execution time that was parallelized.

We can see the plot of the speedup of the tool for a given number of threads in Figure 21 (obtained using 'real') where every line is a combination of gadget and operation (RPS_COR1, RPC_COR2, ...), and it can be seen that the speedup highly depends upon the combination, forming three classes.
To investigate what could prompt such a difference, we can express the program's speedup using Amdahl's law:

$$s_{tot} = \frac{1}{(1-p) + \frac{p}{s_p}}$$

Where p is the fraction of the execution time that benefits from parallelization, while s_p is the speedup of the parallelized fraction.
Due to how the tool was parallelized, the fraction of the execution time that benefits from parallelization can be estimated from the timings as:

$$parallelTime = serialExecution - \frac{T}{T-1}(parallelReal - parallelExecution/T)$$
$$p = parallelTime/serialExecution$$

This is plotted in Figure 22, where we can see that it has a minimal dependency on how many threads it's executed with, and that it highly depends on the combination of the gadget and the operation.

5. Conclusions

This thesis shows how RPC is not a settled concept as various alternative definitions are indeed possible, many with execution time or accuracy advantages, and some even defined in ways that aren't easily re-written with the usual tools of simulatabilty, standard deviation and random experiment, opening the field to many more possibilities.
Both in RPS and RPC, the best of the explored alternatives seem to create an accuracy and execution time trade-off, and the exponential nature of the problem means that this trade-off can span entire order of magnitudes. At the same time, a slower execution that rises the accuracy could also be rewarding, as a lower accuracy may cause an increased production cost due to the need to artificially increase the noise in the microchip to at least reach the required $\max_p : f(p) \leq p$, or the gadget will only make the secret leak faster.
This thesis has also shown that the correlation matrix is effective at expressing in compact ways a multitude of definitions and approximations. This effectiveness is present both in how our tool compares to the existing tools, and in how each part of the expression refers to a specific characteristic of the definition/approximation, with similar parts appearing even across RPS and RPC. It's also effective in showing what kind of information is used by a given approximation, for example the effects of going from an SD-based definition to an all-or-nothing simulation with failure.

References

[1] Miklós Ajtai. Secure computation with information leaking to an adversary. *Electron. Colloquium Comput. Complex.*, TR11, 2011.

[2] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. Cryptology ePrint Archive, Paper 2018/566, 2018. `https://eprint.iacr.org/2018/566`.

[3] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with o(1/ log(n)) leakage rate. Cryptology ePrint Archive, Paper 2016/173, 2016. `https://eprint.iacr.org/2016/173`.

[4] Alberto Battistello, Jean-Sebastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the isw masking scheme. Cryptology ePrint Archive, Paper 2016/540, 2016. `https://eprint.iacr.org/2016/540`.

[5] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. Cryptology ePrint Archive, Paper 2020/786, 2020. `https://eprint.iacr.org/2020/786`.

[6] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. Ironmask: Versatile verification of masking security. Cryptology ePrint Archive, Paper 2021/1671, 2021. `https://eprint.iacr.org/2021/1671`.

[7] Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. Cryptology ePrint Archive, Paper 2021/434, 2021. `https://eprint.iacr.org/2021/434`.

[8] Sonia Belaïd, Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud. Dynamic random probing expansion with quasi linear asymptotic complexity. Cryptology ePrint Archive, Paper 2021/1455, 2021. `https://eprint.iacr.org/2021/1455`.

[9] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. Cryptology ePrint Archive, Paper 2021/880, 2021. `https://eprint.iacr.org/2021/880`.

[10] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: from probing attacks to noisy leakage. Cryptology ePrint Archive, Paper 2014/079, 2014. `https://eprint.iacr.org/2014/079`.

[11] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, 2003.

[12] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Hai-Ning Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021:235–274, 2021.

[13] Maria Chiara Molteni and Vittorio Zaccaria. On the spectral features of robust probing security. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020:24–48, 2020.

[14] Thomas Prest, Dahmun Goudarzi, Ange Martinelli, and Alain Passelègue. Unifying leakage models on a rényi day. Cryptology ePrint Archive, Paper 2019/138, 2019. `https://eprint.iacr.org/2019/138`.

A. On the proof that RPC_SD is RPC-like

The hardest part of the proof that RPC_SD is RPC-like is the composition in series. We report here a relatively simple theorem that is central to that part of the proof, and it works for a fixed set of leaking wires.

SD of composition Given four probabilistic set functions H, h, G, g intended as two pairs of real (G, H) and simulator (g, h) such that:

- the functions toward the output (g, G) preserves the probes of the function toward the input (h, H), which are assumed to be the last p outputs. In particular they can be expressed as the parallels $g = g'|id, G = G'|id$
- The simulator h only receives inputs $\preceq I$ and returns outputs $\preceq M|[1^p]$
- The simulator g' only receives inputs $\preceq M$ and returns outputs $\preceq O$
- The input functions (g, G) have the same type, same for the output functions (h, H).
- It's possible to do the composition in series $I^G = O^H$

Then, if we define the following:

$$\begin{aligned}
\forall k \in \mathbb{F}_2^{I^{G'}}, \qquad & SD_k^{G',g'} = \mathrm{SD}\left[G'(k) \cap O; g'(k \cap M)\right] \\
\forall k \in \mathbb{F}_2^{I^H}, \qquad & SD_k^{H,h} = \mathrm{SD}\left[H(k) \cap M; h(k \cap I)\right] \\
\forall k \in \mathbb{F}_2^{I^H}, \qquad & SD_k^{GH,gh} = \mathrm{SD}\left[G(H(k)) \cap O; g(h(k \cap I))\right]
\end{aligned}$$

We can prove that:

$$\forall k \in \mathbb{F}_2^{I^H}, \quad SD_k^{GH,gh} \leq SD_k^{H,h} + \sum_{w \in \mathbb{F}_2^{I^{G'}}} \Pr\left[H(k)|_{<I^{G'}} = w\right] SD_w^{G',g'}$$

In particular, it's the sum of the SD of the input function and the average of the SD of the output function over its possible inputs and weighted by their probability.

Proof We can define

$$\begin{aligned}
\forall k \in \mathbb{F}_2^{I^H}, \qquad & SD_k^{GH,gH} = \mathrm{SD}\left[G(H(k)) \cap O; g(H(k) \cap M)\right] \\
\forall k \in \mathbb{F}_2^{I^H}, \qquad & SD_k^{gH,gh} = \mathrm{SD}\left[g(H(k) \cap M); g(h(k \cap I))\right]
\end{aligned}$$

Then as SD is a metric we have the triangle inequality:

$$\forall k \in \mathbb{F}_2^{I^H}, \quad SD_k^{GH,gh} \leq SD_k^{GH,gH} + SD_k^{gH,gh}$$

For $SD_k^{gH,gh}$ we can apply lemma 1 (see below) and obtain that:

$$SD_k^{gH,gh} \leq SD_k^{H,h}$$

For $SD_k^{GH,gH}$ we can apply lemma 2 (see below) and obtain that:

$$SD_k^{GH,gH} \leq \sum_{w \in \mathbb{F}_2^{I^{G'}}} \Pr\left[V|_{<I^{G'}} = w\right] SD_w^{G',g'}$$

This once substituted into the result of the triangle inequality give the thesis.
Lemma 1: given a probabilistic vectorial function g and two probabilistic vectors A, B, then:

$$\mathrm{SD}\left[g(A); g(B)\right] \leq \mathrm{SD}\left[A; B\right]$$

The proof is:

$$\begin{aligned}
\mathrm{SD}\left[g(A);g(B)\right] =& \frac{1}{2}\sum_{x\in\mathbb{F}_2^{O^g}} \left|\Pr\left[g(A)=x\right]-\Pr\left[g(B)=x\right]\right| \\
=& \frac{1}{2}\sum_{x\in\mathbb{F}_2^{O^g}} \left|\sum_{y\in\mathbb{F}_2^{I^g}} \Pr\left[A=y\wedge g(y)=x\right]-\sum_{y\in\mathbb{F}_2^{I^g}} \Pr\left[B=y\wedge g(y)=x\right]\right| \\
=& \frac{1}{2}\sum_{x\in\mathbb{F}_2^{O^g}} \left|\sum_{y\in\mathbb{F}_2^{I^g}} \Pr\left[A=y\right]\Pr\left[g(y)=x\right]-\sum_{y\in\mathbb{F}_2^{I^g}} \Pr\left[B=y\right]\Pr\left[g(y)=x\right]\right| \\
=& \frac{1}{2}\sum_{x\in\mathbb{F}_2^{O^g}} \left|\sum_{y\in\mathbb{F}_2^{I^g}} \left(\Pr\left[A=y\right]-\Pr\left[B=y\right]\right)\Pr\left[g(y)=x\right]\right| \\
\leq& \frac{1}{2}\sum_{x\in\mathbb{F}_2^{O^g}}\sum_{y\in\mathbb{F}_2^{I^g}} \left|\Pr\left[A=y\right]-\Pr\left[B=y\right]\right|\Pr\left[g(y)=x\right] \\
=& \frac{1}{2}\sum_{y\in\mathbb{F}_2^{I^g}} \left|\left(\Pr\left[A=y\right]-\Pr\left[B=y\right]\right)\right|\sum_{x\in\mathbb{F}_2^{O^g}}\Pr\left[g(y)=x\right] \\
=& \frac{1}{2}\sum_{y\in\mathbb{F}_2^{I^g}} \left|\left(\Pr\left[A=y\right]-\Pr\left[B=y\right]\right)\right| \\
=& \mathrm{SD}\left[A;B\right]
\end{aligned}$$

Lemma 2: given a probabilistic vector V, two probabilistic vector functions a, b such that there is $id : id(x) = x \wedge a' : a = a'|id \wedge b' : b = b'|id$. Then:

$$\mathrm{SD}\left[a(V);b(V)\right] \leq \sum_{w\in\mathbb{F}_2^{I^{a'}}} \Pr\left[V|_{<I^{a'}}=w\right]\mathrm{SD}\left[a'(w);b'(w)\right]$$

Where $w|_{<k}$ returns the vector with the component with index $< k$, i.e. the first k components of w.
Proof:

$$\begin{aligned}
\mathrm{SD}\left[a(V);b(V)\right] =& \frac{1}{2}\sum_{x\in\mathbb{F}_2^{O^a}} \left|\Pr\left[a(V)=x\right]-\Pr\left[b(V)=x\right]\right| \\
=& \frac{1}{2}\sum_{x\in\mathbb{F}_2^{O^a}} \left|\sum_{y\in\mathbb{F}_2^{I^a}} \Pr\left[V=y\right]\Pr\left[a(y)=x\right]-\sum_{y\in\mathbb{F}_2^{I^g}} \Pr\left[V=y\right]\Pr\left[b(y)=x\right]\right| \\
=& \frac{1}{2}\sum_{x'\in\mathbb{F}_2^{p}}\sum_{x''\in\mathbb{F}_2^{O^{a'}}} \left|\sum_{y'\in\mathbb{F}_2^{p}}\sum_{y''\in\mathbb{F}_2^{I^{a'}}} \Pr\left[V=y'|y''\right]\left(\Pr\left[a'(y')=x'\wedge y''=x''\right]-\Pr\left[b'(y')=x'\wedge y''=x''\right]\right)\right| \\
=& \frac{1}{2}\sum_{x'\in\mathbb{F}_2^{p}}\sum_{x''\in\mathbb{F}_2^{O^{a'}}} \left|\sum_{y'\in\mathbb{F}_2^{p}}\sum_{y''\in\mathbb{F}_2^{I^{a'}}} \Pr\left[V=y'|y''\right]\mathrm{is}\left[y''=x''\right]\left(\Pr\left[a'(y')=x'\right]-\Pr\left[b'(y')=x'\right]\right)\right| \\
\leq& \frac{1}{2}\sum_{x'\in\mathbb{F}_2^{p}}\sum_{x''\in\mathbb{F}_2^{O^{a'}}}\sum_{y'\in\mathbb{F}_2^{p}}\sum_{y''\in\mathbb{F}_2^{I^{a'}}} \Pr\left[V=y'|y''\right]\mathrm{is}\left[y''=x''\right]\left|\Pr\left[a'(y')=x'\right]-\Pr\left[b'(y')=x'\right]\right| \\
=& \frac{1}{2}\sum_{x'\in\mathbb{F}_2^{O^{a'}}}\sum_{y'\in\mathbb{F}_2^{p}}\sum_{y''\in\mathbb{F}_2^{I^{a'}}} \Pr\left[V=y'|y''\right]\left|\Pr\left[a'(y')=x'\right]-\Pr\left[b'(y')=x'\right]\right| \\
=& \sum_{y'\in\mathbb{F}_2^{p}}\sum_{y''\in\mathbb{F}_2^{I^{a'}}} \Pr\left[V=y'|y''\right]\frac{1}{2}\sum_{x'\in\mathbb{F}_2^{O^{a'}}}\left|\Pr\left[a'(y')=x'\right]-\Pr\left[b'(y')=x'\right]\right| \\
=& \sum_{y'\in\mathbb{F}_2^{I^{a'}}}\left(\sum_{y''\in\mathbb{F}_2^{p}} \Pr\left[V=y'|y''\right]\right)\mathrm{SD}\left[a'(y');b'(y')\right] \\
=& \sum_{w\in\mathbb{F}_2^{I^{a'}}} \Pr\left[V|_{<p}=w\right]\mathrm{SD}\left[a'(w);b'(w)\right]
\end{aligned}$$

Abstract in lingua italiana

In questa tesi esploriamo le proprietà di sicurezza contro sonde probabilistiche su un circuito, ovvero come la sua sicurezza resiste un avversario in grado di vedere il valore di ogni filo con una certa probabilità. In particolare, ci focalizziamo sulla Random Probing Security (RPS) e sulla Random Probing Composability (RPC) di un circuito o gadget. Per l' RPS, esploriamo varie approssimazioni, mentre per l'RPC estendiamo la definizione esistente in una classe di definizioni con tre proprietà condivise. In aggiunta alla definizione originale, mostriamo due esempi significativi di definizioni in questa classe. Sia per l'RPS che per l'RPC mostriamo come definizioni e approssimazioni già presenti in letteratura formano espressioni significative quando scritte usando la matrice di correlazione. Da qui, mostriamo che tipo d'informazione sui gadget ognuna di esse usa. Infine, abbiamo creato un tool software per calcolare le varie funzioni usando la matrice di correlazione di un dato gadget generico. Usiamo questo tool su vari gadget per analizzare il compromesso tra tempo di esecuzione e accuratezza, e per confrontare questi risultati con i tool esistenti.

Parole chiave: Side-channel security, Random probing security, Verifica automatica, composizione, trasformata di Walsh, matrice di correlazione

Acknowledgements