Optimizing S-box Implementations for Several Criteria using SAT Solvers

Ko Stoffelen

Radboud University, Digital Security,
Nijmegen, The Netherlands
k.stoffelen@cs.ru.nl

Abstract. We explore the feasibility of applying SAT solvers to optimizing implementations of small functions such as S-boxes for multiple optimization criteria, e.g., the number of nonlinear gates and the number of gates. We provide optimized implementations for the S-boxes used in Ascon, ICEPOLE, Joltik/Piccolo, Keccak/Ketje/Keyak, LAC, Minalpher, PRIMATEs, Prøst, and RECTANGLE, most of which are candidates in the secound round of the CAESAR competition. We then suggest a new method to optimize for circuit depth and we make tooling publicly available to find efficient implementations for several criteria. Furthermore, we illustrate with the 5-bit S-box of PRIMATEs how multiple optimization criteria can be combined.

Keywords: S-box, SAT solvers, implementation optimization, multiplicative complexity, circuit depth complexity, shortest linear straight-line program.

1 Introduction

Implementations of cryptographic algorithms are typically optimized for one or multiple criteria, such as latency, throughput, power consumption, memory consumption, etc., but also criteria such as the cost of adding masking countermeasures to protect against side-channel attacks. It is worthwhile to spend time on this optimization, as the implementations are typically used many times. It is usually a hard problem to find an implementation that is actually theoretically minimal with respect to the criteria, e.g., general circuit minimization is $\sum_2^P$-complete [10]. However, for small functions this is still possible, using, for instance, SAT solvers. Especially for building blocks that can be used in multiple cryptographic algorithms, such as S-boxes, it is useful to look at methods for finding minimal implementations with respect to some given criteria.

In Section 2, we first discuss the simpler problem of finding minimal implementations of linear functions. We give a brief overview of methods for finding the shortest linear straight-line program.

*This work was supported by the European Commission through the Horizon 2020 program under project number ICT-645622 (PQCRYPTO).

We then move towards S-boxes and in Section 3 we consider known methods [13,20] that manage to find minimal implementations for the relevant optimization criteria of multiplicative complexity [9], bitslice gate complexity [12], and gate complexity. The definitions of these criteria are given in Section 3. We study how feasible the methods actually are by applying them to S-boxes that are used in recent cryptographic algorithms, such as several candidates in the CAESAR competition and lightweight block ciphers. Additionally, we provide tools that allow anyone to conveniently do the same to other small S-boxes.

Then we look at another optimization criterion: the circuit depth complexity. This is relevant in hardware implementations to decrease the delay and to be able to increase the clock frequency. We suggest a new method for encoding the circuit depth complexity decision problem in SAT and we show how feasible this method is in practice by providing efficient low-depth S-box implementations for Joltik [17], Piccolo [22], LAC [23], Prøst [18], and RECTANGLE [24] in Section 3.5.

Finally, in Section 4 it is discussed how several optimization criteria can be combined, by first optimizing the S-box used by the PRIMATEs [2] for multiplicative complexity and then for gate complexity. This is done by taking the intermediate result after optimizing for multiplicative complexity, identifying the linear parts of this, and by treating these as instances of the shortest linear straight-line program problem.

Contributions of this paper. To summarize, the contributions of this paper are

- implementations of the S-boxes in Ascon, ICEPOLE, Joltik/Piccolo, Keccak/Ketje/Keyak, LAC, Minalpher, Prøst, and RECTANGLE with a provably minimal number of nonlinear gates;
- a new method for encoding the circuit depth complexity decision problem as an instance of SAT;
- optimized and sometimes even provably minimal implementations of the S-boxes in Joltik/Piccolo, LAC, Prøst, and RECTANGLE with respect to bitslice gate complexity, gate complexity, and circuit depth complexity;
- a method to combine multiple optimization criteria;
- an implementation of the S-box used by the PRIMATEs that is first optimized for multiplicative complexity and then for (bitslice) gate complexity;
- tools and documentation to optimize implementations of small nonlinear functions such as S-boxes using SAT solvers, with respect to multiplicative complexity, bitslice gate complexity, gate complexity, or circuit depth complexity, are put into the public domain. These tools are available online.

2 The Shortest Linear Straight-Line Program Problem

Before tackling the optimization of S-boxes, let us restrict ourselves to linear functions and let us consider the Shortest Linear Program (SLP) problem over

$GF(2)$. Let $\boldsymbol{A}$ be an $m \times n$ matrix of constants over $GF(2)$ and let $\boldsymbol{x}$ be a vector of n variables over $GF(2)$. The SLP problem is to find the program with the smallest number of lines that computes $\boldsymbol{Ax}$, where every program line is of a certain form.

Let Z be a set of variables over $GF(2)$, that initially contains the input variables $\{x_0, \ldots, x_{n-1}\}$. Let $z_i, z_j \in Z$. Then every program line is of the form

$$z' := z_i + z_j.$$

After executing this program line, the new variable z' is added to the set, $Z := Z \cup \{z'\}$. The new variable z' can therefore be used in the next program line. The program is said to compute $\boldsymbol{Ax}$ when $\exists (z_1, \ldots, z_m) \in Z^m \, \{\boldsymbol{Ax} = (z_1, \ldots, z_m)^\intercal\}$ holds.

Being able to find the shortest straight-line linear program has obvious applications to cryptology. Solving the SLP over $GF(2)$ is equivalent to finding the shortest circuit to compute a function using only `XOR` gates. Optimizing implementations of linear operations, such as MixColumns in AES and the linear transformation in certain implementations of SubBytes, can therefore be seen as instances of the SLP problem over $GF(2)$. However, this method does not apply to nonlinear operations such as S-boxes. We show in Section 3 what kind of methods can be used in such cases.

Solving the SLP problem. Boyar, Matthews, and Peralta showed in [7] that the SLP problem over $GF(2)$ is NP-hard. Off-the-shelf SAT solvers can be used to find solutions for small instances of this problem. Fuhs and Schneider-Kamp presented a method [16] to encode the SLP problem as an instance of SAT and they show how this can be used to optimize the affine transformation of AES's SubBytes [15,16].

For larger instances, exact methods will quickly become infeasible. Alternatively, Boyar and Peralta published an approach to solve the SLP problem over $GF(2)$ based on a heuristic [8]. In short, the heuristic method uses a base vector set S, initialized with unit vectors for all variables in $\boldsymbol{x}$, and a distance vector *Dist*[] that keeps track of the minimal Hamming distance to S for each row in $\boldsymbol{A}$. Repeatedly, the sum of the pair of base vectors in S that minimizes the sum of *Dist*[] is added to S and *Dist*[] is updated, until *Dist*[] is the all-zero vector. If there is a tie between two pairs of base vectors, the pair that maximizes the Euclidean length of the new *Dist*[] vector is chosen. This algorithm makes it possible to find solutions to larger instances of the SLP problem.

3 Optimizing S-box Implementations using SAT-solvers

For nonlinear functions such as S-boxes, known approaches based on heuristics [8] all exploit additional algebraic structure that may be available, e.g., as for the S-box of AES. However, in general this additional structure may not exist and one may need to fall back to generic methods such as SAT solvers.

S-box implementations in both software and hardware can be optimized with SAT solvers according to several criteria. In this paper we consider the following optimization goals:

Multiplicative complexity The multiplicative complexity of a function [9] is defined as the smallest number of nonlinear gates with fan-in 2 required to compute this function. If we restrict our S-box implementations to the {`AND`, `OR`, `XOR`, `NOT`} operations, we only need to consider the number of `AND`s and `OR`s. Optimizing for this goal is useful in the case of protecting against side-channel attacks using random masks, where nonlinear gates are typically more expensive to mask. There are also applications in multi-party computation and fully homomorphic encryption, where the cost of nonlinear operations is even more significant [1].

Bitslice gate complexity The bitslice gate complexity of a function [12] is defined as the smallest number of operations in {`AND`, `OR`, `XOR`, `NOT`} required to compute this function. This translates directly to efficient bitsliced software implementations, as on most common CPU architectures, there are no instructions for computing `NAND`, `NOR`, or `XNOR` immediately.

Gate complexity The gate complexity of a function is defined as the smallest number of logic gates required to compute this function. Unlike for bitslice gate complexity, `NAND`, `NOR`, and `XNOR` gates are now also allowed. This translates to efficient hardware implementations, although the different amounts of area required by these types of gates and the different delays still need to be taken into account. Note that we only consider gates with a fan-in of at most 2.

Circuit depth complexity The depth of a circuit is defined as the length of the longest paths from an input gate to an output gate. Every function can be computed by a circuit with depth 2, e.g., by expressing the function in conjunctive or disjunctive normal form. However, this can lead to very wide circuits with a lot of gates, which is typically not desirable. There is somewhat of a trade-off between circuit depth and number of gates. Still, optimizing for this goal is useful in the case of hardware implementations, to be able to decrease the total delay and therefore to be able to increase the clock frequency. Again, only gates with a fan-in of at most 2 are considered.

These criteria come with corresponding decision problems. For example, given a function f and some positive integer k, the **multiplicative complexity decision problem** is defined as:

> "Is there a circuit that implements f and that uses at most k nonlinear operations?"

The decision problems for the other three optimization goals can be defined analogously. Off-the-shelf SAT solvers can be used to solve these decision problems. When a SAT solver successfully finds a circuit for some value k but outputs `UNSAT` for $k-1$, it is proven that k is the minimum value. Note that when a SAT solver outputs `SAT` for some value k, it also provides a satisfying valuation that can be used to reconstruct an implementation of f.

In order to use SAT solvers to solve these decision problems, the problems first have to be encoded in logical formulas in conjunctive normal form (CNF), because that is the input format that the SAT solver requires.

3.1 Notation

For the encoding, we use the notation of [20]. We consider systems of multivariate equations over $GF(2)$. In these equations, let:

- x_i be variables representing S-box inputs;
- y_i be variables representing S-box outputs;
- q_i be variables representing gate inputs;
- t_i be variables representing gate outputs;
- a_i be variables representing wiring between gates;
- b_i be variables representing wiring 'inside' gates. This will become more clear when they are first used in Section 3.3.

In the implementations the *logical connectives* are used to denote the types of operations, i.e., let $\wedge$, $\vee$, $\oplus$, $\neg$ denote `AND`, `OR`, `XOR`, `NOT`, respectively, and let $\uparrow$, $\downarrow$, $\leftrightarrow$ denote `NAND`, `NOR`, `XNOR`, respectively.

3.2 Optimizing for Multiplicative Complexity

Courtois, Mourouzis and Hulme [13,20] suggested a method to encode the multiplicative complexity decision problem. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an S-box and let k be the multiplicative complexity that we want to test for. Then first create a set of equations C in ANF consisting of:

- $\forall i \in \{0, \ldots, k-1\}$: $t_i = q_{2i} \cdot q_{2i+1}$, to encode the k `AND` gates.
- $\forall i \in \{0, \ldots, 2k-1\}$: $q_i = a_l + \left(\sum_{j=0}^{n-1} a_{l+j+1} \cdot x_j\right) + \left(\sum_{j=0}^{\lfloor \frac{i}{2} \rfloor -1} a_{l+n+j+1} \cdot t_j\right)$, where $l = i(n+1) + \left\lfloor \frac{i^2-2i+1}{4} \right\rfloor$, to encode that the inputs of the `AND` gates can be any linear combination of S-box inputs and previous `AND` gate outputs. The single a represents an optional `NOT` gate.
- $\forall i \in \{0, \ldots, m-1\}$: $y_i = \left(\sum_{j=0}^{n-1} a_{s+j} \cdot x_j\right) + \left(\sum_{j=0}^{k-1} a_{s+n+j} \cdot t_j\right)$, where $s = 2k(n+1) + k(k-1) + i(n+k)$, to encode that the S-box outputs can be any linear combination of S-box inputs and `AND` gate outputs.

For example, when $n = m = 4$ and $k = 3$, this leads to the following set of equations C:

$$
\begin{aligned}
q_0 &= a_0 + a_1 \cdot x_0 + a_2 \cdot x_1 + a_3 \cdot x_2 + a_4 \cdot x_3 \\
q_1 &= a_5 + a_6 \cdot x_0 + a_7 \cdot x_1 + a_8 \cdot x_2 + a_9 \cdot x_3 \\
t_0 &= q_0 \cdot q_1 \\
q_2 &= a_{10} + a_{11} \cdot x_0 + a_{12} \cdot x_1 + a_{13} \cdot x_2 + a_{14} \cdot x_3 + a_{15} \cdot t_0
\end{aligned}
$$

$$
\begin{aligned}
q_3 &= a_{16} + a_{17} \cdot x_0 + a_{18} \cdot x_1 + a_{19} \cdot x_2 + a_{20} \cdot x_3 + a_{21} \cdot t_0 \\
t_1 &= q_2 \cdot q_3 \\
q_4 &= a_{22} + a_{23} \cdot x_0 + a_{24} \cdot x_1 + a_{25} \cdot x_2 + a_{26} \cdot x_3 + a_{27} \cdot t_0 + a_{28} \cdot t_1 \\
q_5 &= a_{29} + a_{30} \cdot x_0 + a_{31} \cdot x_1 + a_{32} \cdot x_2 + a_{33} \cdot x_3 + a_{34} \cdot t_0 + a_{35} \cdot t_1 \\
t_2 &= q_4 \cdot q_5 \\
y_0 &= a_{36} \cdot x_0 + a_{37} \cdot x_1 + a_{38} \cdot x_2 + a_{39} \cdot x_3 + a_{40} \cdot t_0 + a_{41} \cdot t_1 + a_{42} \cdot t_2 \\
y_1 &= a_{43} \cdot x_0 + a_{44} \cdot x_1 + a_{45} \cdot x_2 + a_{46} \cdot x_3 + a_{47} \cdot t_0 + a_{48} \cdot t_1 + a_{49} \cdot t_2 \\
y_2 &= a_{50} \cdot x_0 + a_{51} \cdot x_1 + a_{52} \cdot x_2 + a_{53} \cdot x_3 + a_{54} \cdot t_0 + a_{55} \cdot t_1 + a_{56} \cdot t_2 \\
y_3 &= a_{57} \cdot x_0 + a_{58} \cdot x_1 + a_{59} \cdot x_2 + a_{60} \cdot x_3 + a_{61} \cdot t_0 + a_{62} \cdot t_1 + a_{63} \cdot t_2
\end{aligned}
$$

This set of equations does not depend on f yet, but only on the values of n and m. The equations in C have to be satisfied for all possible S-box inputs. An equation set C' is created that contains 2^n copies of the equations in C, in which all x_i, y_i, q_i, t_i are renumbered, but in which all a_i, b_i remain the same. f is 'bound' to the problem description by adding its truth table as $2^n(n+m)$ constant equations, i.e., one for every bit in both the S-box input and the S-box output, to C'.

C' is in ANF. The method by Bard, Courtois, and Jefferson [3] for converting sparse systems of low-degree multivariate polynomials over $GF(2)$ is used to convert C' to CNF, such that it is understood by the SAT solver.

Results. This method makes it feasible to find the multiplicative complexity of several 4-bit and 5-bit S-boxes. Finding the multiplicative complexity comes with an actual implementation that uses this minimal number of nonlinear gates. After Courtois, Hulme, and Mourouzis applied this method to the S-boxes of PRESENT and GOST [12], we show that we can also find results for more recently introduced 4-bit and 5-bit S-boxes.

We consider the S-boxes, and if applicable, their inverses (denoted by $^{-1}$), in Ascon [14], ICEPOLE [19], Keccak [4]/Ketje [5]/Keyak [6], all PRIMATEs [2], Joltik [17]/Piccolo [22], LAC [23], Minalpher [21], Prøst [18], and RECTANGLE [24]. Minalpher's and Prøst's S-boxes are involutory, which is why their inverses are not listed separately. The inverse S-boxes in Ascon, ICEPOLE, Keccak, Ketje, and Keyak are not actually used in decryption and are therefore not considered.

For all S-boxes except the one used by the PRIMATEs we are able to prove the multiplicative complexity. The results are summarized in Table 1. The actual implementations can be found in Appendix A, but note that these should not be used by themselves as we are being very generous with `XOR` gates. The linear parts should be optimized separately, as we will demonstrate in Section 4.

These and subsequent results are obtained using MiniSat 2.2.0[1] and CryptoMiniSat 2.9.10[2] using default parameters on a single core of an Intel Xeon

[1] http://www.minisat.se/MiniSat.html

[2] http://www.msoos.org/cryptominisat2/

S-box	Size nxm	Multiplicative complexity
Ascon	5x5	5
ICEPOLE	5x5	6
Keccak/Ketje/Keyak	5x5	5
PRIMATEs	5x5	$\in \{6, 7\}$
PRIMATEs^{-1}	5x5	$\in \{6, 7, 8, 9, 10\}$
Joltik/Piccolo	4x4	4
Joltik^{-1}/Piccolo^{-1}	4x4	4
LAC	4x4	4
Minalpher	4x4	5
Prøst	4x4	4
RECTANGLE	4x4	4
RECTANGLE^{-1}	4x4	4

Table 1. Multiplicative complexity of S-boxes

E7-4870 v2 running at 2.30 GHz.

For the PRIMATEs S-box and inverse S-box, we find solutions for $k = 7$ and $k = 10$, respectively. Furthermore, we find for both S-boxes that the case for $k = 5$ yields `UNSAT`. We have started several attempts to find a decisive answer for $k = 6$, including

– reducing the CNF, e.g., using NICESAT [11];
– fine-tuning SAT solver parameters;
– trying other SAT solvers;
– trying other SAT solvers that can run in parallel on many cores, such as Plingeling and Treengeling[3]; and
– letting all of this run for several months on a machine with 120 cores and 3 TB of RAM.

Unfortunately, none of these attempts resulted in an answer as no solver instance has terminated yet. As these SAT solvers typically have much more difficulty with proving the `UNSAT` case than proving the `SAT` case, and as the `SAT` proof for $k = 7$ was found in less than 40 hours, we expect the $k = 6$ case to yield `UNSAT` and we therefore conjecture the multiplicative complexity of the PRIMATEs S-box to be 7. In Section 4 we go into more detail on optimizing the PRIMATEs S-box. For the inverse S-box, we did not manage to find solutions for $k \in \{6, 7, 8, 9\}$.

3.3 Optimizing for Bitslice Gate Complexity

In [13,20], a method is also given to optimize for bitslice gate complexity. However, it is only applied on the small CTC2 toy cipher and therefore it remains unclear

[3] http://fmv.jku.at/lingeling/

how practical this method is for real-world ciphers. We investigate this by applying the method to the same S-boxes as in the previous section.

The encoding scheme for the bitslice gate complexity decision problem is slightly different compared to the multiplicative complexity decision problem. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ again be an S-box and let k now be the bitslice gate complexity that we want to test for. Then our first set of equations C in ANF consists of:

- $\forall i \in \{0, \ldots, k-1\}$: $t_i = b_{3i} \cdot q_{2i} \cdot q_{2i+1} + b_{3i+1} \cdot q_{2i} + b_{3i+1} \cdot q_{2i+1} + b_{3i+2} + b_{3i+2} \cdot q_{2i}$, to encode the k `AND`, `OR`, `XOR` or `NOT` gates. The b_i determine what kind of gate this will represent, as can be seen in Table 2.
- $\forall i \in \{0, \ldots, k-1\}$: $0 = b_{3i} \cdot b_{3i+2}$ and $0 = b_{3i+1} \cdot b_{3i+2}$, to make sure that the gate is either a unary `NOT` or a binary `AND`/`OR`/`XOR`, but not the `XOR` of them. This excludes `NAND`/`NOR`/`XNOR` gates.
- $\forall i \in \{0, \ldots, 2k-1\}$: $q_i = \left(\sum_{j=0}^{n-1} a_{l+j} \cdot x_j\right) + \left(\sum_{j=0}^{\lfloor \frac{i}{2} \rfloor - 1} a_{l+n+j} \cdot t_j\right)$, where $l = in + \left\lfloor \frac{i^2-2i+1}{4} \right\rfloor$, to encode that the inputs of the gates can be any S-box input bit or any previously computed bit.
- $\forall i \in \{0, \ldots, 2k-1\}$, $\forall j \in \{l, \ldots, l+n+\lfloor \frac{i}{2} \rfloor - 2\}$, $\forall u \in \{j+1, \ldots, l+n+\lfloor \frac{i}{2} \rfloor - 1\}$: $0 = a_j \cdot a_u$, to encode an 'at most one' constraint on the gate inputs.
- $\forall i \in \{0, \ldots, m-1\}$: $y_i = \left(\sum_{j=0}^{n-1} a_{s+j} \cdot x_j\right) + \left(\sum_{j=0}^{k-1} a_{s+n+j} \cdot t_j\right)$, where $s = 2kn + k(k-1) + i(n+k)$, to encode that the S-box output bit can be any S-box input bit or any gate output.
- $\forall i \in \{0, \ldots, m-1\}$, $\forall j \in \{s, \ldots, s+n+k-2\}$, $\forall u \in \{j+1, \ldots, s+n+k-1\}$: $0 = a_j \cdot a_u$, to encode an 'at most one' constraint on the S-box outputs.

$b_{3i}b_{3i+1}b_{3i+2}$	Gate t_i function
`000`	0
`001`	$\neg q_{2i}$
`010`	$q_{2i} \oplus q_{2i+1}$
`011`	Prevented by constraint on b_{3i+2}
`100`	$q_{2i} \wedge q_{2i+1}$
`101`	Prevented by constraint on b_{3i+2}
`110`	$q_{2i} \vee q_{2i+1}$
`111`	Prevented by constraint on b_{3i+2}

Table 2.

Converting C to C' and then to CNF is the same process as with the multiplicative complexity decision problem. Note that the 'constraint equations' on a_i and b_j do not have to be duplicated 2^n times for C', as they are not renumbered. This saves a lot of redundant clauses.

Results. As the amount of CNF clauses that is necessary to describe the bitslice gate complexity decision problem becomes much larger compared to the multiplicative complexity decision problem, it can take much more time for a SAT solver to actually solve a problem instance. Still, for some 4-bit and 5-bit S-boxes results can be obtained within minutes or within a few hours. Table 3 contains some examples. If a bitslice gate complexity is listed as $\leq k$, a solution was found for k, but we were unable to prove that this is the minimum because the SAT solver did not terminate within a reasonable amount of time for $k-1$. The actual implementations with the given number of operations can be found in Appendix A.

S-box	Size nxm	Bitslice gate complexity	Implementation
Keccak/Ketje/Keyak	5x5	≤ 13	3 `AND`, 2 `OR`, 5 `XOR`, 3 `NOT`
Joltik/Piccolo	4x4	10	1 `AND`, 3 `OR`, 4 `XOR`, 2 `NOT`
Joltik^{-1}/Piccolo^{-1}	4x4	10	1 `AND`, 3 `OR`, 4 `XOR`, 2 `NOT`
LAC	4x4	11	2 `AND`, 2 `OR`, 6 `XOR`, 1 `NOT`
Minalpher	4x4	≥ 11	
Prøst	4x4	8	4 `AND`, 4 `XOR`
RECTANGLE	4x4	$\in \{11, 12\}$	1 `AND`, 3 `OR`, 7 `XOR`, 1 `NOT`
RECTANGLE^{-1}	4x4	$\in \{10, 11, 12\}$	4 `OR`, 7 `XOR`, 1 `NOT`

Table 3. Bitslice gate complexity of S-boxes

For Prøst and the (forward) S-box of RECTANGLE, it is interesting to note that the SAT solvers are able to find the same implementations as the corresponding authors already suggested. We have proven that their bitsliced implementations are indeed minimal.

3.4 Optimizing for Gate Complexity

A method to encode the gate complexity decision problem was also provided in [13,20], but again, actual results were only given for the CTC2 toy cipher. We show that it is feasible to compute the gate complexity for real-world 4-bit S-boxes as well.

The encoding is very similar to the bitslice gate complexity decision problem. The first set of equations C in ANF only differs in two places:

– Instead of the previous rule for t_i, the gates are encoded differently: $\forall i \in \{0, \ldots, k-1\}$: $t_i = b_{3i} \cdot q_{2i} \cdot q_{2i+1} + b_{3i+1} \cdot q_{2i} + b_{3i+1} \cdot q_{2i+1} + b_{3i+2}$, to encode the k gates. The b_i determine what kind of gate this will represent, as can be seen in Table 4.
– The additional constraints on the b_i are completely omitted.

Converting C to C' and then to CNF is similar to the previous optimization goals.

$b_{3i}b_{3i+1}b_{3i+2}$	Gate t_i function
000	0
001	1
010	$q_{2i} \oplus q_{2i+1}$
011	$q_{2i} \leftrightarrow q_{2i+1}$
100	$q_{2i} \wedge q_{2i+1}$
101	$q_{2i} \uparrow q_{2i+1}$
110	$q_{2i} \vee q_{2i+1}$
111	$q_{2i} \downarrow q_{2i+1}$

Table 4.

Results. Our results on real-world 4-bit S-boxes are summarized in Table 5. The full implementations can be found in Appendix A. For our 5-bit S-boxes we did not manage to retrieve results. Note that all types of logic gates are considered equally expensive. There is no type of gate that is preferred over the other, because information such as differences in area consumption or time delay are not taken into account. The implementations found by the SAT solver should therefore not be used directly for hardware implementations. However, they serve as an optimal starting point from where to swap 'expensive' gates for cheaper ones, depending on the specific technology that is to be used. For example, the designers of Piccolo suggested a hardware implementation [22] of their S-box that may or may not be more efficient than the implementation given here, depending on the specific technology.

S-box	Gate complexity	Implementation
Joltik/Piccolo	8	2 `OR`, 1 `XOR`, 2 `NOR`, 3 `XNOR`
Joltik^{-1}/Piccolo^{-1}	8	2 `OR`, 1 `XOR`, 2 `NOR`, 3 `XNOR`
LAC	10	1 `AND`, 3 `OR`, 2 `XOR`, 4 `XNOR`
Prøst	8	4 `AND`, 4 `XOR`
RECTANGLE	$\in \{10, 11\}$	1 `AND`, 1 `OR`, 2 `XOR`, 1 `NAND`, 1 `NOR`, 5 `XNOR`
RECTANGLE^{-1}	$\in \{10, 11\}$	1 `AND`, 1 `OR`, 6 `XOR`, 1 `NAND`, 1 `NOR`, 1 `XNOR`

Table 5. Gate complexity of S-boxes

3.5 Optimizing for Depth Complexity

There are many situations in high-speed hardware implementations where the implementer wants to keep the depth of the circuit as low as possible, in order to be able to increase the clock frequency, without having to use significantly more gates. We provide a novel method to find low-depth implementations of small functions such as S-boxes using SAT solvers. This method is inspired by the encoding of the gate complexity decision problem, but modified in some important ways.

In the encoding of the gate complexity decision problem, we expressed that every gate can use the S-box input and the outputs of previous gates as its input. The key idea here is to divide the circuit into *depth layers* and to encode the notion that a gate can only use the S-box input and the output of gates in the previous layers as its input. This is made more precise later.

First we note that it is necessary to limit the potential increase of the number of gates when reducing the depth of a circuit. We introduce a fixed maximum layer width w to address this, so we allow at most w gates to be executed in parallel. For some function f, we want to be able to answer questions such as: "is there a circuit implementing f with depth k and with at most w gates on each depth layer?".

Using this fixed maximum layer width, we make our encoding method more precise by once more creating a set C of multivariate equations over $GF(2)$ in ANF that consists of:

– $\forall i \in \{0, \ldots, kw-1\}$: $t_i = b_{3i} \cdot q_{2i} \cdot q_{2i+1} + b_{3i+1} \cdot q_{2i} + b_{3i+1} \cdot q_{2i+1} + b_{3i+2}$, to encode the kw gates. The b_i determine what kind of gate this will represent, as can be seen in Table 4.
– $\forall i \in \{0, \ldots, 2kw-1\}$: $q_i = \left(\sum_{j=0}^{n-1} a_{l+j} \cdot x_j\right) + \left(\sum_{j=0}^{v-1} a_{l+n+j} \cdot t_j\right)$, where $v = \left\lfloor \frac{i}{2w} \right\rfloor w$ and $l = in + v\,(i - v - w)$, to encode that the inputs of the gates can be any S-box input bit or any previously computed bit.
– $\forall i \in \{0, \ldots, 2kw-1\}$, $\forall j \in \{l, \ldots, l+n+v-2\}$, $\forall u \in \{j+1, \ldots, l+n+v-1\}$: $0 = a_j \cdot a_u$, to encode an 'at most one' constraint on the gate inputs.
– $\forall i \in \{0, \ldots, m-1\}$: $y_i = \left(\sum_{j=0}^{n-1} a_{s+j} \cdot x_j\right) + \left(\sum_{j=0}^{kw-1} a_{s+n+j} \cdot t_j\right)$, where $s = kw(2n + kw - w) + i(n + kw)$, to encode that the S-box output bit can be any S-box input bit or any gate output.
– $\forall i \in \{0, \ldots, m-1\}$, $\forall j \in \{s, \ldots, s+n+kw-2\}$, $\forall u \in \{j+1, \ldots, s+n+kw-1\}$: $0 = a_j \cdot a_u$, to encode an 'at most one' constraint on the S-box outputs.

Converting C to C' and subsequently expressing this in CNF is again the same process as before.

Results. Using our method, we are able to find low-depth implementations for our 4-bit S-boxes. The results are summarized in Table 6 and the corresponding implementations can be found in Appendix A. The last column in Table 6 lists scenarios that yield `UNSAT`, to show boundaries on what is possible. The trade-off between circuit depth and the number of gates is made here in such a way that reducing the depth by 1 would imply the implementation to have at least twice as many gates as is required by the gate complexity.

4 Combining Criteria: Optimizing the PRIMATEs S-box

So far, we have seen how to optimize for one specific goal. However, a result that is optimized for multiplicative complexity may contain more `XOR` gates than is desired, and a result that is optimized for gate complexity may contain more

S-box	Depth complexity	w	Implementation	`UNSAT` boundaries
Joltik/Piccolo	4	2	2 `OR`, 1 `XOR`,	$k = 4, w = 1$
			2 `NOR`, 3 `XNOR`	$k = 3, w = 10$
Joltik^{-1}/Piccolo^{-1}	4	3	3 `OR`, 5 `XOR`,	$k = 4, w = 2$
			1 `NOR`, 3 `XNOR`	$k = 3, w = 10$
LAC	3	6	3 `OR`, 4 `XOR`,	$k = 3, w = 4$
			4 `NAND`, 4 `XNOR`	$k = 2, w = 10$
Prøst	4	3	4 `AND`, 1 `OR`, 4 `XOR`,	$k = 4, w = 2$
			1 `NAND`, 1 `XNOR`	$k = 3, w = 10$
RECTANGLE	3	6	2 `AND`, 3 `OR`, 5 `XOR`,	$k = 3, w = 4$
			1 `NAND`, 1 `NOR`, 3 `XNOR`	$k = 2, w = 10$
RECTANGLE^{-1}	3	6	1 `OR`, 8 `XOR`,	$k = 3, w = 4$
			3 `NAND`, 2 `NOR`, 2 `XNOR`	$k = 2, w = 10$

Table 6. Depth complexity of S-boxes

nonlinear gates than is desired for a masked implementation. Here we show how multiple optimization goals can be combined by looking at the 5-bit PRIMATEs S-box. We first optimize for multiplicative complexity to have a minimal number of nonlinear gates, and subsequently we minimize the number of linear gates. The result is an implementation that has 4 `AND`, 3 `OR`, 31 `XOR`, and 5 `NOT` gates.

The PRIMATEs S-box is an almost bent permutation with a maximum linear and differential probability of 2^{-4}. It is chosen because of its low area consumption in hardware implementations.

When the optimization method for multiplicative complexity is applied, we find a solution with multiplicative complexity 7 as follows:

$$\begin{aligned}
q_0 &= x_0 \oplus x_3 & q_9 &= x_2 \oplus t_0 \oplus t_3 \\
q_1 &= x_1 & t_4 &= q_8 \wedge q_9 \\
t_0 &= q_0 \vee q_1 & q_{10} &= x_0 \oplus x_3 \oplus x_4 \\
q_2 &= \neg(x_1 \oplus x_3) & q_{11} &= \neg(x_0 \oplus x_4) \\
q_3 &= x_0 \oplus x_2 & t_5 &= q_{10} \vee q_{11} \\
t_1 &= q_2 \wedge q_3 & q_{12} &= \neg(x_1 \oplus x_2 \oplus t_0 \oplus t_2 \oplus t_3 \oplus t_4) \\
q_4 &= x_0 \oplus x_1 \oplus x_4 & q_{13} &= x_2 \oplus x_3 \\
q_5 &= x_0 \oplus x_2 \oplus x_3 & t_6 &= q_{12} \wedge q_{13} \\
t_2 &= q_4 \wedge q_5 & y_0 &= x_1 \oplus x_3 \oplus t_2 \oplus t_3 \oplus t_5 \oplus t_6 \\
q_6 &= \neg(x_0 \oplus x_2 \oplus x_3 \oplus x_4) & y_1 &= x_0 \oplus x_4 \oplus t_1 \oplus t_2 \oplus t_3 \oplus t_4 \oplus t_5 \oplus t_6 \\
q_7 &= x_1 \oplus x_2 \oplus x_4 & y_2 &= x_1 \oplus x_2 \oplus x_4 \oplus t_1 \oplus t_3 \oplus t_4 \oplus t_5 \\
t_3 &= q_6 \vee q_7 & y_3 &= x_0 \oplus x_2 \oplus x_3 \oplus x_4 \oplus t_3 \oplus t_4 \oplus t_5 \oplus t_6 \\
q_8 &= x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 & y_4 &= \neg(x_2 \oplus t_0 \oplus t_2 \oplus t_3 \oplus t_4 \oplus t_5 \oplus t_6)
\end{aligned}$$

It is not hard to see that there are a lot of redundant `XOR` operations in this implementation. We distinguish between `XOR` operations before the nonlinear gates (on x_i) and XOR operations after the nonlinear gates (on t_i). It is possible to see them as two straight-line linear programs, where the first describes the linear part of the S-box approached from the input and the second describes the linear part approached from the S-box output.

The shortest linear straight-line program problem $\boldsymbol{A_1 x_1}$ can be given by

$$\boldsymbol{A_1} = \begin{array}{c} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \\ q_9 \\ q_{10} \\ q_{11} \\ q_{12} \\ q_{13} \\ y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{array} \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{x_1} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}.$$

The shortest linear straight-line program problem $\boldsymbol{A_2 x_2}$ can be given by

$$\boldsymbol{A_2} = \begin{array}{c} q_9 \\ q_{12} \\ y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{array} \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad \boldsymbol{x_2} = \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \end{pmatrix}.$$

We are able to find a minimal straight-line program computing $\boldsymbol{A_2 x_2}$ using SAT solvers. We use the method suggested by Fuhs and Schneider-Kamp [16] to encode the SLP problem as a SAT instance in CNF. This yields a result that is incorporated in our implementation of the PRIMATEs S-box. Finding a minimal straight-line program computing $\boldsymbol{A_1 x_1}$ turned out to be infeasible using SAT solvers within a reasonable amount of time. Therefore, we apply the heuristic approach as suggested by Boyar and Peralta [8]. This does provide us with a short straight-line program. We combine both results and amend the original

PRIMATEs S-box implementation to get the more efficient implementation below, where z_i represent helper variables.

$$\begin{aligned}
z_0 &= x_0 \oplus x_4 & q_7 &= x_4 \oplus z_1 & z_5 &= t_2 \oplus z_4 \\
z_1 &= x_1 \oplus x_2 & t_3 &= q_6 \vee q_7 & z_6 &= t_1 \oplus t_6 \\
z_2 &= x_2 \oplus x_3 & q_8 &= q_4 \oplus z_2 & z_7 &= t_4 \oplus z_5 \\
q_0 &= x_0 \oplus x_3 & z_9 &= t_0 \oplus t_3 & z_8 &= t_1 \oplus z_7 \\
t_0 &= q_0 \vee x_1 & q_9 &= x_2 \oplus z_9 & z_{10} &= t_0 \oplus z_7 \\
q_2 &= x_1 \oplus x_3 & t_4 &= q_8 \wedge q_9 & z_{11} &= t_4 \oplus z_4 \\
q_3 &= \neg(x_0 \oplus x_2) & q_{10} &= \neg(x_3 \oplus z_0) & z_{12} &= z_6 \oplus z_{11} \\
t_1 &= q_2 \vee q_3 & t_5 &= q_{10} \wedge z_0 & y_0 &= \neg(q_2 \oplus z_5) \\
q_4 &= x_1 \oplus z_0 & q_{12} &= \neg(z_1 \oplus z_9 \oplus t_2 \oplus t_4) & y_1 &= z_0 \oplus z_8 \\
q_5 &= x_0 \oplus z_2 & t_6 &= q_{12} \wedge z_2 & y_2 &= q_7 \oplus z_{12} \\
t_2 &= q_4 \wedge q_5 & z_3 &= t_5 \oplus t_6 & y_3 &= q_6 \oplus z_{11} \\
q_6 &= \neg(x_4 \oplus q_5) & z_4 &= t_3 \oplus z_3 & y_4 &= x_2 \oplus z_{10}
\end{aligned}$$

We are able to decrease the previous result of 58 `XOR` gates to only 31 `XOR` gates.

Tools. We provide tools to generate C' in ANF for all discussed optimization goals and to convert a SAT solver solution back to an S-box implementation. We place those tools into the public domain. They and additional documentation are available online at https://github.com/Ko-/sboxoptimization.

5 Conclusion

SAT solvers can be used to find minimal implementations for small functions such as S-boxes with respect to criteria as the multiplicative complexity, bitslice gate complexity, gate complexity, and circuit depth complexity. We have shown how this can be done and how multiple criteria can be combined. However, for 8-bit S-boxes and larger functions these methods quickly become infeasible. One will then have to resort to approaches based on heuristics.

References

1. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer Berlin Heidelberg, 2015.
2. Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1.02. CAESAR submission: http://competitions.cr.yp.to/round2/primatesv102.pdf, 2015. http://primates.ae/.

3. Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. IACR Cryptology ePrint Archive, Report 2007/024, 2007. http://eprint.iacr.org/.
4. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche. The Keccak reference, January 2011. http://keccak.noekeon.org/.
5. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles van Assche, and Ronny van Keer. Ketje v1. CAESAR submission: http://competitions.cr.yp.to/round1/ketjev11.pdf, 2014. http://ketje.noekeon.org/.
6. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles van Assche, and Ronny van Keer. Keyak v2. CAESAR submission: http://competitions.cr.yp.to/round2/keyakv2.pdf, 2015. http://keyak.noekeon.org/.
7. Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In *Mathematical Foundations of Computer Science 2008*, volume 5162 of *Lecture Notes in Computer Science*, pages 168–179. Springer Berlin Heidelberg, 2008.
8. Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer Berlin Heidelberg, 2010.
9. Joan Boyar, René Peralta, and Denis Pochuev. On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science*, 235(1):43–57, 2000.
10. David Buchfuhrer and Christopher Umans. The complexity of Boolean formula minimization. In *Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 24–35. Springer Berlin Heidelberg, 2008.
11. Benjamin Chambers, Panagiotis Manolios, and Daron Vroon. Faster SAT solving with better CNF generation. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 1590–1595, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.
12. Nicolas Courtois, Daniel Hulme, and Theodosis Mourouzis. Solving circuit optimisation problems in cryptography and cryptanalysis. Cryptology ePrint Archive, Report 2011/475, 2011. http://eprint.iacr.org/.
13. Nicolas Courtois, Theodosis Mourouzis, and Daniel Hulme. Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits. *International Journal On Advances in Intelligent Systems*, 6(3 and 4):165–176, 2013.
14. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.1. CAESAR submission: http://competitions.cr.yp.to/round2/asconv11.pdf, 2015. http://ascon.iaik.tugraz.at.
15. Carsten Fuhs and Peter Schneider-Kamp. Optimizing the AES S-box using SAT. In *IWIL@ LPAR*, pages 64–70. Citeseer, 2010.
16. Carsten Fuhs and Peter Schneider-Kamp. Synthesizing shortest linear straight-line programs over GF(2) using SAT. In *Theory and Applications of Satisfiability Testing – SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 71–84. Springer Berlin Heidelberg, 2010.
17. Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Joltik v1.3. CAESAR submission: http://competitions.cr.yp.to/round2/joltikv13.pdf, 2015. http://www1.spms.ntu.edu.sg/~syllab/m/index.php/Joltik.
18. Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst v1.1. CAESAR submission: http://competitions.cr.yp.to/round1/proestv11.pdf, 2014.

19. Paweł Morawiecki, Kris Gaj, Ekawat Homsirikamol, Krystian Matusiewicz, Josef Pieprzyk, Marcin Rogawski, Marian Srebrny, and Marcin Wójcik. ICEPOLE v2. CAESAR submission: http://competitions.cr.yp.to/round2/icepolev2.pdf, 2015.
20. Theodosis Mourouzis. *Optimizations in Algebraic and Differential Cryptanalysis*. PhD thesis, UCL (University College London), 2015.
21. Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. Minalpher v1.1. CAESAR submission: http://competitions.cr.yp.to/round2/minalpherv11.pdf, 2015.
22. Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer Berlin Heidelberg, 2011.
23. Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, and Jian Zhang. LAC: A lightweight authenticated encryption cipher. CAESAR submission: http://competitions.cr.yp.to/round1/lacv1.pdf, 2014.
24. Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: A bit-slice ultra-lightweight block cipher suitable for multiple platforms. Cryptology ePrint Archive, Report 2014/084, 2014. http://eprint.iacr.org/.

A Optimized S-box Implementations

For all given implementations, x_0 and y_0 denote the most significant bit of the S-box input x and the S-box output y, respectively.

A.1 Optimized for Multiplicative Complexity

Only implementations that do not reach the minimal number of nonlinear operations when optimizing for other criteria are listed here. The implementations below serve as a demonstration of what kind of output can be expected from SAT solvers when optimizing for multiplicative complexity. To increase the amount of solutions and therefore the likelihood that we will find one fast, we do not put restrictions on the number of linear gates, which is why the implementations below are not very efficient. The number of linear gates can be reduced further as shown in Section 4.

Ascon
k = 5

$q_0 = \neg(x_3 \oplus x_4)$

$q_1 = \neg x_4$

$t_0 = q_0 \wedge q_1$

$q_2 = x_0 \oplus x_2 \oplus x_4$

$q_3 = x_1$

$t_1 = q_2 \wedge q_3$

$q_4 = x_0 \oplus x_1 \oplus x_4$

$q_5 = x_1$

$t_2 = q_4 \wedge q_5$

$q_6 = x_3 \oplus x_4$

$q_7 = x_0$

$t_3 = q_6 \wedge q_7$

$q_8 = x_3 \oplus t_1 \oplus t_2$

$q_9 = x_1 \oplus x_2$
$t_4 = q_8 \wedge q_9$
$y_0 = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus t_1$
$y_1 = x_0 \oplus x_2 \oplus x_3 \oplus x_4 \oplus t_4$
$y_2 = x_1 \oplus x_2 \oplus x_3 \oplus t_0$
$y_3 = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus t_3$
$y_4 = x_3 \oplus x_4 \oplus t_2$

ICEPOLE
k = 6

$q_0 = x_0 \oplus x_3 \oplus x_4$
$q_1 = x_0 \oplus x_3$
$t_0 = q_0 \wedge q_1$
$q_2 = \neg(x_2 \oplus x_4)$
$q_3 = x_2 \oplus x_3 \oplus x_4$
$t_1 = q_2 \wedge q_3$
$q_4 = x_2 \oplus t_0 \oplus t_1$
$q_5 = x_0 \oplus x_2 \oplus x_3 \oplus x_4 \oplus t_1$
$t_2 = q_4 \wedge q_5$
$q_6 = x_0 \oplus x_1 \oplus x_4$
$q_7 = x_1 \oplus x_4$
$t_3 = q_6 \wedge q_7$
$q_8 = x_1 \oplus x_2 \oplus t_0 \oplus t_1 \oplus t_2$
$q_9 = x_0 \oplus x_1 \oplus t_0 \oplus t_1 \oplus t_2$
$t_4 = q_8 \wedge q_9$
$q_{10} = \neg(x_2 \oplus t_1 \oplus t_3 \oplus t_4)$
$q_{11} = \neg(x_0 \oplus t_4)$
$t_5 = q_{10} \wedge q_{11}$
$y_0 = x_0 \oplus t_0 \oplus t_1 \oplus t_2 \oplus t_5$
$y_1 = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus t_2 \oplus \cdots$
$\cdots \oplus t_3 \oplus t_4 \oplus t_5$
$y_2 = x_0 \oplus x_3 \oplus t_1 \oplus t_2 \oplus t_3 \oplus t_4 \oplus t_5$
$y_3 = x_0 \oplus x_2 \oplus x_3 \oplus x_4 \oplus t_0 \oplus t_1 \oplus \cdots$
$\cdots \oplus t_2 \oplus t_3 \oplus t_4 \oplus t_5$
$y_4 = x_2 \oplus x_4 \oplus t_0 \oplus t_1 \oplus t_2 \oplus t_4 \oplus t_5$

PRIMATEs
k = 7
See Section 4.

PRIMATEs^{-1}
k = 10

$q_0 = x_0 \oplus x_2 \oplus x_3$
$q_1 = \neg(x_2 \oplus x_4)$
$t_0 = q_0 \wedge q_1$
$q_2 = x_0$
$q_3 = x_1$
$t_1 = q_2 \wedge q_3$
$q_4 = x_2 \oplus x_3 \oplus t_0$
$q_5 = \neg x_1$
$t_2 = q_4 \wedge q_5$
$q_6 = x_1 \oplus t_1 \oplus t_2$
$q_7 = x_2 \oplus x_4$
$t_3 = q_6 \wedge q_7$
$q_8 = x_2 \oplus t_0 \oplus t_2 \oplus t_3$
$q_9 = x_0 \oplus x_3 \oplus x_4 \oplus t_1 \oplus t_2 \oplus t_3$
$t_4 = q_8 \wedge q_9$
$q_{10} = x_0 \oplus x_2 \oplus x_3 \oplus t_1 \oplus t_2 \oplus t_3$
$q_{11} = x_1 \oplus x_3 \oplus t_0 \oplus t_2$
$t_5 = q_{10} \wedge q_{11}$
$q_{12} = x_0 \oplus x_4$
$q_{13} = t_0 \oplus t_3 \oplus t_4 \oplus t_5$
$t_6 = q_{12} \wedge q_{13}$
$q_{14} = \neg(x_0 \oplus x_1 \oplus x_2 \oplus x_4 \oplus t_0 \oplus \cdots$
$\cdots \oplus t_1 \oplus t_3 \oplus t_4 \oplus t_5 \oplus t_6)$
$q_{15} = x_0 \oplus x_3 \oplus t_0 \oplus t_1 \oplus t_2 \oplus t_4 \oplus t_6$
$t_7 = q_{14} \wedge q_{15}$
$q_{16} = \neg(x_2 \oplus x_3 \oplus t_2 \oplus t_5)$
$q_{17} = \neg(x_0 \oplus x_1 \oplus x_4 \oplus t_0 \oplus t_1 \oplus \cdots$
$\cdots \oplus t_2 \oplus t_3 \oplus t_6 \oplus t_7)$
$t_8 = q_{16} \wedge q_{17}$
$q_{18} = x_4 \oplus t_2 \oplus t_5 \oplus t_6 \oplus t_8$
$q_{19} = \neg(x_0 \oplus x_1 \oplus x_4 \oplus t_4 \oplus t_7 \oplus t_8)$
$t_9 = q_{18} \wedge q_{19}$

$y_0 = x_0 \oplus x_1 \oplus t_0 \oplus t_6 \oplus t_7 \oplus t_9$
$y_1 = t_0 \oplus t_3 \oplus t_6$
$y_2 = t_3 \oplus t_5 \oplus t_6 \oplus t_7$
$y_3 = t_1 \oplus t_2 \oplus t_4$
$y_4 = x_1 \oplus t_0 \oplus t_4 \oplus t_8$

Minalpher
k = 5

$q_0 = x_1 \oplus x_2 \oplus x_3$
$q_1 = x_1$
$t_0 = q_0 \wedge q_1$
$q_2 = x_0 \oplus x_1 \oplus x_3$
$q_3 = x_1 \oplus x_2 \oplus t_0$
$t_1 = q_2 \wedge q_3$
$q_4 = x_0 \oplus t_0$
$q_5 = x_0 \oplus x_1 \oplus x_2 \oplus t_0$
$t_2 = q_4 \wedge q_5$
$q_6 = \neg(x_0 \oplus x_1 \oplus x_2 \oplus t_0 \oplus t_2)$
$q_7 = \neg(x_0 \oplus x_1 \oplus t_1)$
$t_3 = q_6 \wedge q_7$
$q_8 = x_0 \oplus x_2 \oplus x_3 \oplus t_0 \oplus t_1 \oplus t_2 \oplus t_3$
$q_9 = x_1 \oplus x_2 \oplus t_0 \oplus t_2 \oplus t_3$
$t_4 = q_8 \wedge q_9$
$y_0 = x_2 \oplus t_4$
$y_1 = x_0 \oplus x_2 \oplus t_1$
$y_2 = t_0 \oplus t_3$
$y_3 = t_1 \oplus t_2 \oplus t_3$

A.2 Optimized for Bitslice Gate Complexity

Keccak/Ketje/Keyak
k = 13

$t_0 = \neg x_2$
$t_1 = t_0 \wedge x_3$
$y_1 = t_1 \oplus x_1$
$t_3 = \neg x_4$
$t_4 = t_3 \wedge x_0$
$y_3 = x_3 \oplus t_4$
$t_6 = x_3 \vee t_3$
$y_2 = t_0 \oplus t_6$
$t_8 = \neg x_0$
$t_9 = y_1 \vee t_0$
$t_{10} = t_8 \wedge x_1$
$y_0 = t_9 \oplus t_8$
$y_4 = x_4 \oplus t_{10}$

Joltik/Piccolo
k = 10

$t_0 = x_0 \vee x_1$
$t_1 = t_0 \oplus x_3$
$y_0 = \neg t_1$
$t_3 = x_2 \vee y_0$
$y_2 = t_3 \oplus x_1$
$t_5 = x_1 \vee x_2$
$t_6 = t_5 \oplus x_0$
$t_7 = t_1 \wedge t_6$
$y_3 = x_2 \oplus t_7$
$y_1 = \neg(t_6)$

Joltik^{-1}/Piccolo^{-1}
k = 10

$t_0 = \neg x_1$
$t_1 = \neg x_0$
$t_2 = t_1 \wedge t_0$
$y_2 = t_2 \oplus x_3$
$t_4 = x_0 \vee y_2$
$y_1 = x_2 \oplus t_4$
$t_6 = y_2 \vee y_1$
$y_0 = t_6 \oplus t_0$
$t_8 = y_0 \vee y_1$
$y_3 = t_8 \oplus t_1$

LAC
k = 11

$t_0 = x_3 \oplus x_2$
$t_1 = x_1 \vee x_0$
$y_3 = t_1 \oplus t_0$
$t_3 = x_1 \wedge y_3$
$t_4 = \neg x_3$
$t_5 = t_4 \oplus t_3$
$y_2 = t_5 \oplus x_0$
$t_7 = t_5 \wedge y_2$
$t_8 = y_3 \vee y_3$
$y_1 = t_7 \oplus x_1$
$y_0 = t_8 \oplus x_0$

Prøst
k = 8

$t_0 = x_2 \wedge x_1$
$y_1 = t_0 \oplus x_3$
$t_2 = x_0 \wedge x_1$
$y_0 = x_2 \oplus t_2$
$t_4 = y_1 \wedge y_0$

$y_2 = x_0 \oplus t_4$

$t_6 = y_1 \wedge y_2$

$y_3 = x_1 \oplus t_6$

RECTANGLE
k = 12

$t_0 = x_3 \vee x_0$

$t_1 = x_1 \oplus t_0$

$y_1 = x_2 \oplus t_1$

$t_3 = x_3 \wedge t_1$

$t_4 = x_0 \oplus t_3$

$y_2 = y_1 \oplus t_4$

$t_6 = x_3 \oplus x_2$

$t_7 = \neg y_2$

$t_8 = t_7 \vee t_1$

$y_0 = t_8 \oplus t_6$

$t_{10} = t_7 \vee y_0$

$y_3 = t_{10} \oplus t_1$

RECTANGLE^{-1}
k = 12

$t_0 = \neg x_2$

$t_1 = x_0 \vee t_0$

$t_2 = x_3 \oplus t_1$

$y_2 = t_2 \oplus x_1$

$t_4 = t_0 \vee t_2$

$t_5 = x_0 \oplus t_4$

$y_3 = t_5 \oplus y_2$

$t_7 = t_2 \vee y_3$

$t_8 = t_7 \oplus t_5$

$y_0 = t_8 \oplus x_2$

$t_{10} = y_0 \vee y_3$

$y_1 = t_{10} \oplus t_2$

A.3 Optimized for Gate Complexity

Joltik/Piccolo
k = 8

$t_0 = x_1 \vee x_0$

$t_1 = x_1 \downarrow x_2$

$y_0 = x_3 \leftrightarrow t_0$

$y_1 = t_1 \oplus x_0$

$t_4 = y_1 \vee y_0$

$t_5 = y_0 \downarrow x_2$

$y_2 = t_5 \leftrightarrow x_1$

$y_3 = t_4 \leftrightarrow x_2$

Joltik^{-1}/Piccolo^{-1}
k = 8

$t_0 = x_1 \downarrow x_0$

$y_2 = t_0 \oplus x_3$

$t_2 = y_2 \downarrow x_0$

$y_1 = x_2 \leftrightarrow t_2$

$t_4 = y_1 \vee y_2$

$y_0 = t_4 \leftrightarrow x_1$

$t_6 = y_0 \vee y_1$

$y_3 = t_6 \leftrightarrow x_0$

LAC
k = 10

$t_0 = x_2 \leftrightarrow x_3$

$t_1 = x_1 \wedge t_0$

$t_2 = t_1 \oplus x_3$

$y_2 = x_0 \leftrightarrow t_2$

$t_4 = x_0 \vee x_1$

$y_3 = t_4 \leftrightarrow t_0$

$t_6 = t_3 \vee y_3$

$t_7 = x_0 \vee t_2$

$y_0 = t_6 \oplus x_0$

$y_1 = x_1 \leftrightarrow t_7$

Prøst
k = 8

$t_0 = x_2 \wedge x_1$

$y_1 = t_0 \oplus x_3$

$t_2 = x_0 \wedge x_1$

$y_0 = x_2 \oplus t_2$

$t_4 = y_1 \wedge y_0$

$y_2 = x_0 \oplus t_4$

$t_6 = y_1 \wedge y_2$

$y_3 = x_1 \oplus t_6$

RECTANGLE
k = 11

$t_0 = x_3 \downarrow x_0$

$t_1 = x_1 \oplus t_0$

$t_2 = x_2 \leftrightarrow x_0$

$y_1 = t_1 \leftrightarrow x_2$

$t_4 = t_1 \wedge t_2$

$t_5 = y_1 \leftrightarrow x_3$

$t_6 = t_1 \vee x_3$

$y_2 = t_2 \oplus t_6$

$t_8 = y_2 \uparrow t_5$

$y_3 = t_1 \leftrightarrow t_8$

$y_0 = t_5 \leftrightarrow t_4$

RECTANGLE^{-1}
k = 11

$t_0 = x_3 \vee x_2$

$t_1 = x_0 \oplus t_0$

$t_2 = t_1 \downarrow x_1$

$t_3 = t_2 \oplus x_3$

$y_1 = x_2 \oplus t_3$

$t_5 = t_1 \oplus x_2$

$y_3 = t_5 \oplus x_1$

$t_7 = y_1 \uparrow y_3$

$t_8 = y_3 \wedge t_1$

$y_0 = t_7 \leftrightarrow t_1$

$y_2 = t_8 \oplus t_3$

A.4 Optimized for Depth Complexity

The extra whitespace separates the different depth layers.

Joltik/Piccolo
$\mathbf{k = 4, w = 2}$

$$\begin{aligned} t_0 &= x_1 \lor x_0 \\ t_1 &= x_1 \downarrow x_2 \end{aligned}$$

$$\begin{aligned} y_0 &= x_3 \leftrightarrow t_0 \\ y_1 &= t_1 \oplus x_0 \end{aligned}$$

$$\begin{aligned} t_4 &= y_1 \lor y_0 \\ t_5 &= y_0 \downarrow x_2 \end{aligned}$$

$$\begin{aligned} y_2 &= t_5 \leftrightarrow x_1 \\ y_3 &= t_4 \leftrightarrow x_2 \end{aligned}$$

Joltik^{-1}/Piccolo^{-1}
$\mathbf{k = 4, w = 3}$

$$\begin{aligned} t_0 &= x_1 \lor x_0 \\ t_1 &= x_2 \leftrightarrow x_0 \\ t_2 &= x_3 \oplus x_1 \end{aligned}$$

$$\begin{aligned} t_3 &= t_2 \oplus t_0 \\ y_2 &= x_3 \leftrightarrow t_0 \\ t_5 &= t_0 \oplus t_1 \end{aligned}$$

$$\begin{aligned} t_6 &= t_3 \downarrow t_5 \\ t_7 &= y_2 \lor t_1 \\ t_8 &= y_2 \lor x_0 \end{aligned}$$

$$\begin{aligned} y_1 &= x_2 \oplus t_8 \\ y_3 &= x_0 \oplus t_6 \\ y_0 &= t_3 \leftrightarrow t_7 \end{aligned}$$

LAC
$\mathbf{k = 3, w = 6}$

$$\begin{aligned} t_0 &= x_0 \uparrow x_1 \\ t_1 &= x_3 \oplus x_0 \\ t_2 &= x_3 \leftrightarrow x_2 \\ t_3 &= x_2 \oplus x_0 \\ t_4 &= x_2 \lor x_0 \\ t_5 &= x_1 \lor x_0 \end{aligned}$$

$$\begin{aligned} y_3 &= t_5 \leftrightarrow t_2 \\ t_7 &= t_0 \leftrightarrow t_4 \\ t_8 &= t_5 \uparrow x_3 \\ t_9 &= t_3 \uparrow t_2 \\ t_{10} &= t_5 \lor t_2 \\ t_{11} &= x_1 \uparrow t_2 \end{aligned}$$

$$\begin{aligned} y_1 &= t_{10} \oplus t_7 \\ y_2 &= t_{11} \oplus t_1 \\ y_0 &= t_9 \leftrightarrow t_8 \end{aligned}$$

Prøst
$\mathbf{k = 4, w = 3}$

$$\begin{aligned} t_0 &= x_1 \land x_2 \\ t_1 &= x_1 \land x_0 \\ t_2 &= x_3 \land x_0 \end{aligned}$$

$$\begin{aligned} y_1 &= t_0 \oplus x_3 \\ t_4 &= t_2 \oplus x_1 \\ y_0 &= x_2 \oplus t_1 \end{aligned}$$

$$\begin{aligned} t_6 &= y_0 \uparrow y_1 \\ t_7 &= y_1 \land x_2 \\ t_8 &= t_4 \lor t_2 \end{aligned}$$

$$\begin{aligned} y_2 &= x_0 \leftrightarrow t_6 \\ y_3 &= t_7 \oplus t_8 \end{aligned}$$

RECTANGLE
$\mathbf{k = 3, w = 6}$

$$\begin{aligned} t_0 &= x_0 \downarrow x_3 \\ t_1 &= x_1 \oplus x_2 \\ t_2 &= x_3 \leftrightarrow x_2 \\ t_3 &= x_0 \land x_1 \\ t_4 &= x_1 \land x_2 \\ t_5 &= x_1 \oplus x_0 \end{aligned}$$

$$\begin{aligned} t_6 &= t_4 \lor t_2 \\ t_7 &= x_3 \uparrow t_5 \\ t_8 &= t_4 \oplus t_3 \\ t_9 &= t_1 \lor t_5 \\ y_1 &= t_0 \leftrightarrow t_1 \\ t_{11} &= t_0 \lor t_2 \end{aligned}$$

$$\begin{aligned} y_3 &= t_9 \oplus t_6 \\ y_2 &= t_1 \leftrightarrow t_7 \\ y_0 &= t_8 \oplus t_{11} \end{aligned}$$

RECTANGLE^{-1}
$\mathbf{k = 3, w = 6}$

$$\begin{aligned} t_0 &= x_0 \oplus x_1 \\ t_1 &= x_0 \uparrow x_2 \\ t_2 &= x_3 \leftrightarrow x_2 \\ t_3 &= x_2 \downarrow x_3 \\ t_4 &= x_2 \oplus x_1 \\ t_5 &= x_1 \lor x_0 \end{aligned}$$

$$\begin{aligned} t_6 &= t_3 \oplus x_2 \\ t_7 &= t_3 \downarrow x_1 \\ t_8 &= t_0 \uparrow t_2 \\ t_9 &= t_4 \oplus t_1 \\ t_{10} &= t_4 \uparrow t_1 \\ t_{11} &= t_2 \oplus t_5 \end{aligned}$$

$$\begin{aligned} y_1 &= t_7 \oplus t_{11} \\ y_2 &= t_9 \oplus x_3 \\ y_3 &= t_0 \leftrightarrow t_6 \\ y_0 &= t_{10} \oplus t_8 \end{aligned}$$