1 Introduction

This note specifies two instances of a hash function obtained from applying the Marvellous design strategy [1] to a specific context. The context in question is native hashing in a STARK [2] Virtual Machine such as Miden [9].

This context induces unique design constraints, which this specification addresses. The hash function must be defined over the same field that the VM is defined over, which is the prime field with \( p = 2^{64} - 2^{32} + 1 \) elements. One of the main use cases is Merkle tree hashing, and so the hash function must support an interface for efficient two-to-one hashing. There are two parameter sets, targeting security level 128 and 160, respectively.

2 Specification

The starting point is Rescue-Prime [8], and we assume the reader is familiar with this document. What is described here is the deviations from this standard. A complete reference implementation in SageMath serves as a companion to this specification. It is available at https://github.com/ASDiscreteMathematics/rpo.

2.1 Integer Parameters

Table 1 fixes some integer parameters. Additionally, this choice for \( p \) fixes \( \alpha \) and \( \alpha^{-1} \), which are the exponents of the power maps in the forward and backward S-box layer, respectively (see Fig. 1). Specifically, \( \alpha = 7 \) and

\[
\alpha^{-1} = 10540996611094048183 = 1001001001001001001001100111011011110110110110110111_2.
\]
Table 1: Integer parameters for the two instances of Rescue-Prime Optimized

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Instance 1</th>
<th>Instance 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>prime field modulus $p$</td>
<td>$2^{64} - 2^{32} + 1$</td>
<td>$2^{64} - 2^{32} + 1$</td>
</tr>
<tr>
<td>security level $\lambda$</td>
<td>128</td>
<td>160</td>
</tr>
<tr>
<td>round number $N$</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>state size $m$</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>rate $r$</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>capacity $c$</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

2.2 Round Constants

The round constants are defined as follows:

- Start from the string $\text{RP0(}x_1, x_2, x_3, x_4\text{)}$.
- Populate the wildcards “$x_1$” with the ASCII decimal expansion of the integer parameters $p, m, c, \lambda$, in that order.
- Use SHAKE256 to expand this ASCII string into $9 \cdot 2 \cdot N \cdot m$ pseudorandom bytes.
- For every chunk of 9 bytes, compute the matching integer by interpreting the byte array as as the integer’s base-256 expansion with least significant digit first.
- Reduce the obtained integer modulo $p$.
- Collect all such integers. The list of obtained field elements constitutes the list of round constants.

The function `get_round_constants` of the reference implementation accomplishes this task.

2.3 MDS Matrix

The MDS matrix is circulant. Its first row is

$$[7, 23, 8, 26, 13, 10, 9, 7, 6, 22, 21, 8]$$

for 128 bits of security, and

$$[256, 2, 1073741824, 2048, 16777216, 128, 8, 16, 524288, 4194304, 1, 268435456, 1, 1024, 2, 8192]$$

for 160.

2.4 Order of Operations within a Round

The operations within every half-round are reordered. The correct order is now:

1. MDS matrix
2. injection of constants
3. alpha or alpha-inverse S-box layer.
2.5 Padding Rule

The padding rule makes a distinction depending on whether the length of the input is a multiple of the rate $r$ or not.

- The zero-length input is not allowed.
- If the input length is a multiple of the rate $r$, then
  - Initialize all capacity elements to 0.
  - No changes are made to the message.
- If the input length is not a multiple of the rate $r$, then
  - Initialize the first capacity element to 1 and all others to 0.
  - Append to the input a single 1 element followed by as many zeros as are necessary to make the input length a multiple of the rate.

In either case the sponge methodology applies: absorb $r$ elements from the input in between applications of the permutation, until no input is left.

2.6 Overwrite Mode

In the absorb phase of the sponge construction, the state elements associated with the rate are overwritten by the matching elements from the input chunk, rather than added into. Specifically, if the state elements are $s[0]$ through $s[m-1]$ and the input chunk is $i[0]$ through $i[r-1]$ then correct absorption is given by $s[j] \leftarrow i[j]$ for all $0 \leq j < r$ rather than $s[j] \leftarrow s[j] + i[j]$ for all $0 \leq j < r$.

2.7 Indexation of State Elements

The state is divided into the capacity part, with indices 0 through $c-1$, and the rate part, with indices $c$ through $m-1$. After the last permutation is done, the digest is given by elements $c$ through $c + r/2 - 1$.

3 Test Vectors

The test vectors are generated by the method $\text{print\_test\_vectors}$ from the reference implementation. For the sake of completeness, they are repeated here.
### 3.1 128 Bits Instance

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

### 3.2 160 Bits Instance

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

---

4
4 Motivation

4.1 Circulant MDS Matrix

Rescue-Prime is secure when instantiated with any MDS matrix. Therefore, a circulant MDS matrix such as the one proposed by Polygon Zero may be chosen instead of the one defined by the specification. The obvious questions are

1) Is the proposed matrix really MDS?

2) Does the structure enable faster computation of matrix-vector products?

4.1.1 MDS Test

As for question (1), we were granted access to Polygon Zero’s MDS test procedure and were able to verify the correctness of this function. As a result, we are confident that the circulant matrices specified above are in fact MDS.

In the interest of disseminating science, we present a self-contained description of this algorithm here. Credit for this function goes to Hamish Ivey-Law from Polygon Zero [7].

The key insight is that the standard cofactor expansion method for computing determinants induces a dependency relation on intermediate computation results. The resulting graph is directed and acyclic. As a consequence, it allows for a dynamic programming approach.

Specifically, the dependency relation is this: to compute the determinant of a $(k+1) \times (k+1)$ matrix, you need to select a row or column and combine it with the $(k+1)$-many $k \times k$ determinants of minors that do not cover that row or column. Therefore, if you have the determinants of all $k \times k$ submatrices, computing the determinants of all $(k+1) \times (k+1)$ submatrices is straightforward.

This observation suggests the following strategy: Compute the determinants of all $2 \times 2$ submatrices. Use those values to compute the determinants of all $3 \times 3$ submatrices. And so on until the final dimension has been reached. For every computed determinant, verify that its value is nonzero.

This method tests a matrix for hyperinvertibility, which is the property that every square submatrix is invertible. Hyperinvertibility is equivalent to MDS. The next lemma and proof are folklore knowledge.

**Lemma 1.** A matrix is hyperinvertible iff it is MDS.
Proof. Hyperinvertibility $\Rightarrow$ MDS. Let $M$ be an $m \times m$ hyperinvertible matrix. If there is nonzero a codeword $(\mathbf{x}| \mathbf{x}M)$ whose Hamming weight is less than $m + 1$, then $\text{HW}(\mathbf{x}M) < m + 1 - \text{HW}(\mathbf{x})$. Let $\bar{\mathbf{x}}$ denote $\mathbf{x}$ after dropping the zeros, and $\bar{M}$ the $\text{HW}(\mathbf{x}) \times m$ matrix whose corresponding rows are dropped. Clearly, $\bar{\mathbf{x}}M = \mathbf{x}M$ and so $\text{HW}(\mathbf{x}M) < m + 1 - \text{HW}(\mathbf{x})$. Equivalently, $\bar{\mathbf{x}}M$ has to contain at least $\text{HW}(\mathbf{x})$ zeros. Therefore, some $\text{HW}(\mathbf{x}) \times \text{HW}(\mathbf{x})$ submatrix of $\bar{M}$ and of $\bar{M}$ sends $\bar{\mathbf{x}}$ to 0, which can only happen if it is singular, contradicting the assumption that $M$ is hyperinvertible.

MDS $\Rightarrow$ hyperinvertibility. Let $\mathcal{C} \subset \mathbb{F}^{2m}$ be an MDS code of dimension $m$ and $(I|M)$ its systematic generator matrix. Then $\mathcal{C} = \{(\mathbf{x}| \mathbf{x}M) | \mathbf{x} \in \mathbb{F}^m \}$. If $M$ is not hyperinvertible, then some square $k \times k$ submatrix $\bar{M}$ is singular. Then build a codeword as follows. Set $\mathbf{x}$ to a (left-) kernel vector of $\bar{M}$ in those coordinates that correspond to $\bar{M}$, and to 0 in other coordinates. The codeword $(\mathbf{x}| \mathbf{x}M) \in \mathcal{C}$ has at least $m - k$ zeros in the first half and at least $k$ zeros in the second half, so its Hamming weight is at most $m$, contradicting the assumption of $\mathcal{C}$ being MDS.

To find a suitable circulant MDS matrix, it suffices to sample the first row from a suitable distribution, test the resulting circulant matrix for hyperinvertibility, and repeat if it does not pass the test.

4.1.2 Fast Multiplication

As for question (2), the answer is in the affirmative. Let $R_p = \mathbb{Z}_p[X]/(X^m - 1)$ be the ring of polynomials with multiplication modulo $X^m - 1$. There is an isomorphism between the elements of $R_p$ and circulant $m \times m$ matrices over $\mathbb{Z}_p$ given by

$$a_0 + a_1X + a_2X^2 + \cdots + a_{m-1}X^{m-1} \leftrightarrow \begin{pmatrix} a_0 & a_{m-1} & \cdots & a_1 \\ a_1 & a_0 & & a_2 \\ \vdots & & \ddots & \vdots \\ a_{m-1} & a_{m-2} & \cdots & a_0 \end{pmatrix}.$$  \hspace{1cm} (1)

A fast way to multiply polynomials modulo $X^m - 1$ translates to a fast circulant matrix times vector multiplication procedure. We describe below two methods for fast polynomial multiplication modulo $X^m - 1$.

Note that the coefficient vector of the polynomial corresponds to the first column of the matrix, and not the first row. To translate between row and column, one needs to reverse the entire vector except for the element at the first position.

4.1.3 Karatsuba-based

Karatsuba multiplication [6] splits the multiplication of two polynomials of degree at most $n - 1$ up into three multiplications of polynomials of degree $n/2 - 1$, and applies that split recursively. In the limit the procedure requires only $O(n^{1.58})$ multiplications, compared to the $n^2$ for the schoolbook algorithm. While the number of multiplications is reduced, the number of additions is increased. However, additions generally do not need to be followed up with modular reduction.

Let $a(X) = a_l(X) + X^{n/2} \cdot a_r(X)$ and $b(X) = b_l(X) + X^{n/2} \cdot b_r(X)$ with all of $a_l(X)$, $a_r(X)$, $b_l(X)$, and $b_r(X)$ having degree at most $n/2 - 1$. Let

- $c_0(X) = a_l(X) \times b_l(X)$,
- $c_2(X) = a_r(X) \times b_r(X)$,
- $c_1(X) = (a_l(X) + a_r(X)) \times (b_l(X) + b_r(X)) - c_0(X) - c_2(X)$.
then \( c(X) = a(X) \times b(X) = c_0(X) + X^{n/2} \cdot c_1(X) + X^n \cdot c_2(X) \). Note that \( c(X) \) can be calculated with only 3 multiplications of polynomials of half the number of coefficients. Applying this reduction recursively is what generates Karatsuba’s multiplication algorithm.

After using Karatsuba to find the product of two polynomials \( t(X) \) and \( s(X) \) that represent the MDS matrix and state vector respectively, the next step is the reduction modulo \( X^m - 1 \). This step is straightforward: just iterate over the coefficients of monomials \( X^{m/2} \) through \( X^{m-1} \) and add them to the coefficients of monomials \( X^0 \) through \( X^{m/2-1} \).

### 4.1.4 NTT-based

Using the same isomorphism as in the previous section and the fact that multiplication in the ring \( R_p \) can be done efficiently using the Number Theory Transform (NTT) and its inverse, we can get an \( O(n \cdot \log(n)) \) algorithm for circulant matrix times vector multiplication procedure. More precisely, let \( \omega \) be a primitive \( n \)-th root of unity and define the following \( \mathbb{Z}_p[X] \)-linear map:

\[
\text{NTT}_\omega : \left\{ \begin{array}{l}
R_p \rightarrow (\mathbb{Z}_p)^n \\
\mathbb{R}(X) \mapsto (a(\omega^0), a(\omega^1), \ldots, a(\omega^n))
\end{array} \right.
\]

which is just the evaluation of the polynomial \( a \) at the powers of \( \omega \), i.e. the \( n \)-th roots of unity. This is called the Number Theory Transform (NTT), and is a special case of the Discrete Fourier Transform. Then, using the Chinese Remainder Theorem, one can show that \( \text{NTT}_\omega \) is an isomorphism of algebras. This means that, in particular, the following holds:

\[
\text{NTT}_\omega (a(X) \times b(X)) = \text{NTT}_\omega (a(X)) \odot \text{NTT}_\omega (b(X))
\]

or equivalently

\[
a(X) \times b(X) = \text{NTT}_\omega^{-1}(\text{NTT}_\omega (a(X)) \odot \text{NTT}_\omega(b(X)))
\]

where \( \odot \) is the Hadamard product. This in particular yields the following algorithm:

**Algorithm 1** Circulant matrix times vector multiplication using NTT

**Require:** \( n \geq 1, a(X), b(X) \in R_p \) and \( \omega \) is a primitive \( n \)-th root of unity.

**Ensure:** \( C(X) = a(X) \times b(X) \)

\[
\begin{align*}
\alpha &\leftarrow \text{NTT}_\omega (a(X)) \\
\beta &\leftarrow \text{NTT}_\omega (b(X)) \\
\gamma &\leftarrow \alpha \odot \beta \\
C(X) &\leftarrow \text{NTT}_\omega^{-1}(\gamma)
\end{align*}
\]

Given that in our current context the matrix we are multiplying with is fixed once and for all, the previous algorithm can be optimized by pre-computing the NTT of the MDS matrix such that in the end it will be necessary to compute only one NTT and one inverse NTT per input vector \( b \). Since the NTT and the inverse NTT can be computed in \( O(n \cdot \log(n)) \) using the Fast Fourier Transform (FFT), the complexity of Alg. 1 is also \( O(n \cdot \log(n)) \).

### 4.2 Reduced Round Number

According to the specification [8], the Gröbner basis attack dominates for the range in which the given parameters lie. Moreover, the number of rounds should be set to \( N = \lceil 1.5 \cdot \min(l_1, 5) \rceil \), where \( l_1 \) is the minimal number required to guarantee that the Gröbner basis attack has complexity at least as large as the security parameter, and where the factor 1.5 is a security margin.
For both the 128-bit variant and the 160-bit variant, the estimate for the complexity of the Gröbner basis attack exceeds the security level as soon as $N \geq 3$. Plugging this data point into the formula gives rise to a recommended number of rounds of $N = \lceil 1.5 \cdot 5 \rceil = 8$. Setting instead $N = 7$ constitutes a 12.5% reduction in the number of rounds.

The reason why we feel confident recommending $N = 7$ is threefold:

- The relative reduction is still less than the relative security margin induced by the factor 1.5.

- The estimate of the Gröbner basis attack complexity according to the specification is still more than double the security level. The estimate according to experiments run in the course of this research project were lower, but still indicate that the security target is met with margin.

- Since the publication of the original Marvellous paper [1], the first version of which appeared online in 2019, there has been little progress in attacking either Rescue or Rescue-Prime.

### 4.3 Order of Operations for Better Folding

One of the costly steps in both the prover and the verifier of a STARK is the computation of the vector of values of AIR transition constraint polynomials for two consecutive rows of the algebraic execution trace (AET). The AIR constraints are evaluated point by point on the codewords that represent the trace. The generated codewords are different from zero except in locations that correspond to a row in the AET and its successor.

Rescue and Rescue-Prime have S-boxes that send $x$ to $x^\alpha$, where $\alpha$ is the smallest invertible non-trivial power map degree. To avoid a very high degree AIR, the Marvellous paper [1] introduced folding. This technique involves arithmetizing the evolution of the state forwards in time for a part of the time step, and backwards in time for the remaining part. By traversing over the $x \mapsto x^\frac{1}{\pi}$ map backwards, the degree of the resulting AIR drops to $\alpha$. The equation is found by equating two distinct expressions for the value of the same wire in the middle. The corresponding polynomial is found by moving all terms to the left hand side.

![Figure 2: New versus old folding strategy.](image)

The original folding strategy makes no distinction between the cost of multiplying a vector by a matrix or by its inverse. The AIR polynomials have the same number of terms. However, considerable effort was spent making the MDS matrix-vector multiplication fast, and it seems
difficult to simultaneously make multiplication by the matrix’s inverse fast. This problem mo-
tivates an alternative folding strategy, namely one that avoids using the inverse MDS matrix
altogether.

The new folding strategy in forwards direction: one MDS, one injection of constants, one
\(x \mapsto x^\alpha\) S-box layer, another MDS, and another injection of constants. Only the map \(x \mapsto x^2\) is
computed in backwards direction.

To argue why this re-arrangement does not affect security, consider moving the MDS matrix
and injection of constants of the very last step to the front. This move does not degrade security
according to the following heuristic argument. An attack that meaningfully distinguishes the
new permutation (that is, after the move) from a permutation selected uniformly at random, can
be translated to an attack on the old permutation (that is, before rearrangement) with a linear
overhead. Note that the round constants are sampled independently from those of Rescue-Prime.

4.4 Usability in a Stack-Based Virtual Machine

4.4.1 Indexation

As per § 2.7, the capacity elements are indexed 0 through \(c - 1\) and the rate elements \(c\) through
\(m - 1\). This choice stands in contrast to traditional indexing choices, which puts the rate part
first. While the choice of indexation is irrelevant from a security point of view (see below), this
present choice benefits usability in the context of a stack machine.

The unorthodox indexing scheme corresponds to putting the capacity part deep into the
stack, and the rate part in the shallow end. As a result, squeezing and absorbing corresponds to
operations that affect only the top of the stack. The capacity part of the sponge state does not
need to be touched except when the hasher is initialized and after hashing is finished.

To see why any choice of indexation is arbitrary from the point of view of security, observe
that any permutation can be absorbed into one or even all MDS matrices without changing the
fact that they are MDS. The end result is a permutation with an identical security argument.

4.4.2 Overwrite Mode

In many stack-based VMs, at most one element can be pushed onto the stack per clock cycle.
Therefore, first pushing elements onto the stack and then adding them into the sponge state
would require a large number of operations: in addition to push and add operations, stack
manipulation operations are also necessary to arrange the stack correctly. In the overwrite
mode, we can drop rate elements from the stack and then simply push the new rate elements
onto the stack. This procedure is strictly more efficient than the procedure when elements are
absorbed through addition as it requires no additional stack manipulation operations.

The security of overwrite mode has been analyzed e.g. in § 4.3 of the Sponge SoK [3].

4.5 Padding Rule

A sponge function is defined to absorb \(r\) elements from the input in between applications of the
permutation. When the message cannot be split into an integral number of \(r\)-element blocks,
a padding rule is used to determine how to handle the last block. To avoid trivial collisions, a
padding rule must be sponge compliant (see [3, Def. 1]). The Marvellous paper [1] extended “the
simplest padding rule” (see [3, Def. 2]) suggesting to append a single 1 element at the end of
the message, followed by the necessary amount of zeros to make the length a multiple of \(r\). As
a result, the padded message is always longer than the message before padding.
We are interested in an efficient two-to-one hashing as a primary optimization goal. Therefore, we would like to avoid the costly overhead of an additional permutation call to accommodate the padding. Inspired by [5] and similar to [4] we abuse the notion of domain separation.

We consider messages with length a multiple of $r$ as belonging to the 0-domain, and all other messages as belonging to the 1-domain. As messages in the 0-domain have a predetermined length that is already a multiple of $r$ elements, no further padding is required. For messages in the 1-domain, we apply “the simplest padding rule”. The domain is encoded in the first capacity element which results in a 1-bit security loss because now the attacker has two domains in which to search for “valid” preimages.

Acknowledgments This project was made possible through the financial support of Polygon, for which the authors would like to express gratitude.

References


