Security Analysis of XHASH8/12

Léo Perrin[0]

perrin.leo@gmail.com

Abstract

We have investigated both the padding scheme and the applicability of algebraic attacks to both XHash8 and XHash12. The only vulnerability of the padding scheme we can find is plausibly applicable only in the multi-rate setting—for which the authors make no claim—and is safe otherwise.

For algebraic attack relying on the computation and exploitation of a Gröbner basis, our survey of the literature suggests to base a security argument on the complexity of the variable elimination step rather than that of the computation of the Gröbner basis itself. Indeed, it turns out that the latter complexity is hard to estimate—and is sometimes litteraly non-existent. Focusing on the elimination step, we propose a generalization of the "FreeLunch" approach which, under a reasonable conjecture about the behaviour of the degree of polynomial ideals of dimension 0, is sufficient for us to argue that both XHash8 and XHash12 are safe against such attacks.

We implemented a simplified version of the generation (and resolution) of the corresponding set of equations in SAGE, which allowed us to validate our conjecture at least experimentally, and in fact to show that the lower bound it provides on the ideal degree is not tight—meaning we are a priori understimating the security of these permutations against the algebraic attacks we consider.

At this stage, if used as specified, these hash functions seem safe from Gröbner bases-based algebraic attacks.

Contents

[0]This work was done on behalf of 3milabs.

Notations.

- $p = 2^{64} - 2^{32} + 1$ the field size.
- $\mathbb{F}_p$ the field with p elements. Elements of $\mathbb{F}_q$ are called "words".
- r the rate (size of the outer part), expressed in words.
- c the capacity (size of the inner part), expressed in words.
- ℓ is the number of inverse power maps applied in each round.
- π_8 the full XHASH8 permutation ($\ell = 8$).
- π_{12} the full XHASH12 permutation ($\ell = 12$).

1 On the Padding

1.1 Description

In a sponge-based hash function, the usual approach consists in appending a "1" to the message, and then to add enough zeroes to the result in order to obtain a new message with a length that is a multiple of the rate r. This ensures that two distinct messages will yield two different digests, which would not be the case if only zeroes were added as m and $m||0$ would otherwise yield the same hash. However, in this case, if the initial message is already of a length that is a multiple of r, then hashing it requires processing yet another block, meaning one more call to the permutation. The problem is well known, and has already been discussed e.g. in [Hir16].

The approach used in XHash is similar, in that zeroes are also added to obtain a message of the correct length, without adding any "1". As a consequence, in order to distinguish m from $m||0$, the internal state of the sponge is initialized differently: instead of receiving an all-zero state, the first word of the capacity is set to a value that depends on the congruence modulo r of the message length. While this is not stated in the specification [AKM23], the values used could also depend on the rate itself in order to ensure domain separation between sponge instances using different rates. As it is, XHash does not take the rate directly into account.

More formally, the hashing mode of the XHash family operates as follows on a message $m = (m_0, ..., m_{t-1})$ of t words of $\mathbb{F}_q$.

Initialization. The initial state is initialized as $x = \mathsf{StateInit}(t \mod r)$.

Padding. The message is padded with 0 to obtain a message $m' = (m_0, ..., m_{t-1}, 0, ..., 0)$ of length $t' \geq t$, where the number of 0 is the smallest such that t' is a multiple of r.

Absorption. For $i \in \{0, ..., t'/r)$, do $x = \pi\left(m'_{ri}, ..., m'_{ri+r-1}, x_r, x_{r+1}, ..., x_{11}\right)$, where π is either the XHash8 or XHash12 permutation.

Squeezing. The first 4 words of the outer part are output as the digest.

It is crucial (but easy to ensure) that StateInit has no collision. Below, we assume that it is indeed collision-free.

1.2 On its Security for a Fixed Rate

While there is no formal proof for the security of this mode at this stage, it is easy to check that its specifics have no impact on the usual generic attacks.

Collision Seach. The best attack consists in absorbing random messages until we find two that match in the inner part. We can then asborb identical messages to force an identical content in the rate as well, meaning that the full internal states are then identical. The complexity is roughly the square root of the space corresponding to the capacity, and as we can see the specifics of the initialization have not appeared in the description of this attack.

Preimage. The preimage search also relies on finding a collision in the capacity, except that we go forward (as in the collision search), and backwards (from the targeted digest) using that π is a permutation. Again, the specifics of the initialization do not matter.

1.3 A Potential Issue

In XHash, the rate r is fixed. However, should it be allowed to vary, then the current implementation[1] of Statelnit could be a problem. Indeed, in this case, Statelnit($t \mod r$) is a state of 12 words, namely $(t \mod r, 0, 0, ..., 0)$. The inner part of the state consists in the first $12 - r$ words, and the outer part in the last r ones.

As a consequence, it is possible to find collisions accross instance of XHash that use different rates, i.e. in the multi-rate model [GJMG11]. For instance, we can use the rate/message pairs shown in Table 1, where 0 corresponds to a zero that was present in the state before the absorption of the message. After the absorption of each message by its corresponding sponge instance, the states will match.

r	m	C	State after absorption
4	$(1, 2, 3, 4)$	0	$(C, 0, 0, ..., 0, 1, 2, 3, 4)$
5	$(0, 1, 2, 3, 4)$	0	$(C, 0, ..., 0, 0, 1, 2, 3, 4)$

Table 1: *Building multi-rate collisions through the padding.*

Possible Mitigations. Avoiding this problem can be done in several ways.

1. Ensure that the rate is not to be changed under any circumstance.
2. Use another formula for the computation of C to add a direct dependency in the rate, e.g. replace $C(t) = t \mod r$ with $C(r, t) = r2^4 + (t \mod r)$. Since the state size is of 12 words, this prevents collisions.

[1] https://github.com/0xPolygonMiden/crypto/blob/next/src/hash/rescue/rpx/mod.rs#L178

2 On Algebraic Attacks

2.1 State-of-the-Art

Arguing security against algebraic attacks is a complicated task as these are not so well understood at this stage, and the interplay between the "algebraic" part and more classical techniques can sometimes be used to an attacker's advantage to lower the overall complexity of the attack.

By "algebraic" attack, we mean attacks that ultimately culminate with the resolution of a (system of) equation(s), of which the roots need to be found. Univariate techniques can sometimes be applied, but it does not seem relevant here:

1. the degree of the inverse function is close to the maximum possible, meaning that the univariate degree in any linear combination of the inputs will very quickly become unusable; and
2. the field size ($p = 2^{64} - 2^{32} + 1$) is not that large, meaning that a univariate approach would need to be more efficient than a simple brute-force on one word: a hard task here.

It then leads us to focus on multivariate approaches, i.e. to write a *system* of multivariate equations, that we would then need to solve. Using a terminology inspired by the one introduced in [BBL+24], we can divide these attacks into several steps, as follows.

SysGen. First, the system of equation needs to be generated. Several heuristics are available to this end, but all of them have to introduce new variables whenever a d-th root is used as it is the compositional inverse of these operations that is of a low degree. This step can further be simplified using techniques from "classical" symmetric cryptanalysis, typically based on the probablity one propagation of some affine spaces. This was used to shave off two SPN rounds generically in [BBLP22], with the conditions that the S-boxes are monomials over the base field considered,[2] and that the cipher/permutation starts with an S-box layer. Similar tricks were deployed against Griffin and Arion in [BBL+24], but could not be leveraged against Anemoi.

The system that is generated in the end is not uniquely defined: different generation strategies will yield different systems. For instance, using affine spaces to simplify it will remove some equations. Similarly, we can prefer to introduce new variables and equations in order to get more equations of a lower degree. The existence of an efficient SysGen procedure is implied by arithmetization-orientation, but it is not necessarily the approach used in attacks.[3]

Note that the algorithms used in the next steps are better understood in the case where the system is expected to have a unique (or just a few) solutions. This further adds constraints for the SysGen step, but they are easily handled: the expected number of solutions is easily estimated assuming e.g. that the hash function behaves like a random function, and the input can be contrained for instance by forcing it to be in a vector space of the appropriate dimension.

GröbFind. Once a system is obtained, it is necessary first to endow it with a structure that will allow us to work with it. This in particular allows us to do the

[2] In contrast to what is done in XHash, where monomials are applied over both $\mathbb{F}_p$ and $\mathbb{F}_p^3$.

[3] In fact, our encoding for XHash12 will be basically the STARK one, while the one for XHash8 is significantly different.

polynomial arithmetic we need, e.g. to reduce large degree polynomials modulo lower degree ones. The equations we have obtained define an ideal: since we investigate their common roots, any linear combination will also have these roots. An ideal of polynomials has a *Gröbner basis*, a particular set of polynomials that essentially allow us to properly define a reduction modulo this ideal.

A Gröbner basis is defined for a given *monomial ordering*. Several of these are well known. In particular, a Gröbner basis in *lexicographic* order can greatly simplify the next steps. On the other hand, the complexity of finding a Gröbner basis is highly dependent on said ordering. In general, in order to obtain a basis e.g. in *grevlex* order given any ideal, we need to use either F4 or F5 [Fau99, Fau02]. Unfortunately, efficient open source implementations are hard to find.

VarElim. Once a Gröbner basis is known, we use its structure to extract a univariate polynomial in one of the variables. This extraction step is usually done by reordering the Gröbner basis, i.e. by obtaining a Gröbner basis for a different monomial order, one that is suitable for this purpose. It is usually the lexicographic one. This change of order can be done using the FGLM [FGLM93] algorithm, whose complexity is precisely known as it boils down to linear polynomial arithmetic. It depends on a quantity called the *degree of the ideal* (D_I), and corresponds to the number of roots the system has in the algebraic closure of the field considered. This number is much, much higher than the number of solutions in the field itself (typically 0, 1 or 2).

More custom approaches are sometimes possible; for instance, the authors of [BBL+24] introduced the combination of MatMul and PolyDet: the idea is to bypass the cost of a full FGLM run by focusing on a single variable that is of particular interest in their case.

UniSolve. By design, we expect a solution to exist. Thus, once a univariate equation is extracted, we solve it using well known techniques to get a first root of the system. We then substitute its value in the other equations, and deduce an assignment for all the variables. This works provided that the equations have the correct structure, but this is the case for an output of FGLM (under some reasonable assumptions that have been found to hold in practice when attacking AOPs).

2.2 Constructing a Security Claim Against Root Finding

What would happen if we tried to build a security claim against rootfinding attacks against each of the steps described above?

SysGen. As we established during the discussion above, heuristics exist to lower the complexity of this step, however, it is also possible to mitigate them using appropriate counter-measures (such as starting with a linear diffusion layer). Furthermore, as this step cannot be avoided, it could make sense to base a security claim on the complexity of this step. However, this complexity is hard to estimate, and (as shown in [BBL+24]) tends to be much lower than other steps of the attack. Besides, running for instance a preimage search multiple times for different preimages would lead to systems that are essentially identical, except for some constants in the very last equations. As a consequence, the cost of this step could be ammortized over several attacks.

In the end, we claim that relying on this step for a security bound would not lead to meaningful results.

GröbFind. Experimental results often indicated that this step was the longest in the attack against an AOP, which lead the authors of several primitives to follow those of Rescue [AAB+20], and base their security claims using an estimate of the complexity of F4 or F5 (e.g. Griffin [GHR+23], Anemoi [BBC+23] and Arion [RST23]).

While there is no reason to challenge these experimental results, such claims are now unfortunately falling short. Indeed, it turns out that it is possible to find monomial orderings such that the system obtained during the SysGen step is immediately a Gröbner basis. To the best of our knowledge, this was first put forward to investigate the AES [BPW06], and then adapted to the cryptanalysis of AOPs by two independent teams investigating different algorithms: the authors of [BBL+24] described the "freelunch" approach, which they applied to Anemoi, Arion, XHash8 and in particular Griffin; while Steiner very recently put forward papers presenting such orderings for Poseidon [Ste24b] and Rescue [Ste24a].

VarElim. The complexity of FGLM is well known and "stable": while the complexity of GröbFind can only be upperbounded (while we would need a lower bound anyway), that of FGLM is tight. Furthermore, even though the technique presented in [BBL+24] is more efficient than FGLM, its complexity has a similar structure. Variants of FGLM exist that can be applied e.g. to sparser systems. The applicability of these variants is not so clear, but their complexity also depends on the same quantity: the ideal degree D_I.

We will thus base our security argument on the complexity of this step.

UniSolve. The complexity of this step is tightly known, and is negligible compared to all the other steps considered here. It cannot serve as the basis for a security claim.

2.3 Modeling XHash

In what follows, letters x, y denote variables in $\mathbb{F}_p$, and Greek letters denote multivariate polynomials of $\mathbb{F}_p$. The inner operations of XHash are denoted with upper case Latin letters.

General Approach. For both instances of XHash, we use a similar approach. In order to ensure that the system is expected to have a single solutions (or at most a few), we need to restrict the input to a vector space of dimension r, r being the rate. The simplest approach consists simply in starting with variables $x_0, ..., x_{r-1}$ for the input of the permutation, setting the others to 0.

Then we construct polynomial constraints ensuring that the internal state at the output of the permutation is indeed the image of the initial state after the relevant operations. Passing through low degree operations is easy as we simply need to directly update the polynomials by composing them with said operations. For the (partial) layer of 7th root, we proceed differently: we introduce variables y_j^i just after each 7-th roots, i being the round index and j the word index.

First Round. For the first round, we simply need to encode that the inputs of each 7th root are obtained by applying two MDS layers and a layer of small monomials (along with the appropriate round constants). We deduce 12 equations (resp. 8) for XHash12 (resp. XHash8) of the following shape:

$$(y_j^1)^7 \;=\; C_1 + L_j\left(S\left(L\left(C_0 + \vec{x}\right)\right)\right) \;=\; \alpha_j(x)\ .$$

In order to look for solutions of the CICO problem—or indeed for preimages, we need to fix the capacity words to 0. To this end, we consider that $\vec{x} = (x_0, ..., x_{r-1}, 0, ..., 0)$.

Middle Rounds. We get equations of the following types for the middle rounds of XHash12:

$$(y_j^{i+1})^7 \;=\; C_i + L_j\left(S\left(L\left(C_i + T(y^i)\right)\right)\right) \;=\; \mu_j^i(y^i)\;. \tag{1}$$

Since the layer of small S-boxes (S) and the layer of big S-boxes (T) are both of degree 7, and since the MDS layer L ensures that all outputs depend on all inputs, we have that μ^i is always of degree $7^2 = 49$.

For XHash8, the situation is similar except that we introduce fewer such variables (only 8 per rounds), and that the polynomial constraints take as input the variables from all the previous rounds, not only the immediate predecessor. It complicates writing a closed formula for these constraints, but they can be generated recursively starting from the first one with a computer program that updates the expression of the polynomials corresponding to the "identity" boxes in the high degree layer and keeps those in memory. We then get equations of the form

$$(y_j^{i+1})^7 \;=\; \overline{\mu}_j^i(y^i, y^{i-1}, ..., y^1, x)\;, \tag{2}$$

which are obtained by applying μ^i to a vector whose j-th coordinate is y_j^i if there is a 7-th root at that position, and $\overline{\mu}_j^{i-1}(y^{i-1}, ..., x)$ otherwise. As for μ^i, the polynomials $\overline{\mu}^i$ are of degree 49 in y^i. However, their degree in the previous variables is higher since each of those went through operations of degree 49 during each round.

Final Rounds. For the final round, we do not introduce any new variable. However, we add one equation per word whose value needs to be set to a specific value. Essentially, we get several affine combinations of the image under the big S-box of the y_j^r and (for XHash8) some complex polynomials in all the previous variables which we denote ω_i.

Bypassing Rounds. A natural attack angle at this stage is to try and simplify the system using the knowledge of its structure, a trick usually achieved by carefully tracking the propagation of some affine spaces. This was applied with some success to several SPNs in [BBLP22], and even more so against both Griffin and Arion [BBL+24] due to the specifics of their non-linear layer—in particular, the fact that the non-linear layer of Griffin is affine over large affine spaces.

However, we could not find such heuristics here. In the first round, the first MDS layer restricts the control needed to apply the technique from [BBLP22]. The fact that the non-linear layer is of very high degree everywhere[4] then prevents the applicability of the technique from [BBL+24]. The monomial-based non-linear layer could have potentially lead to the existence of chains of subspaces, as in [BCP23], but the dense and structure-less round constants are an effective counter-measure against it.

Furthermore, the use of the monomial over $\mathbb{F}_p^3$ in the big S-box layer makes it complicated to play with the algebraic representation of several rounds by tightly interweaving its 3 input variables.

Overall, while the threat from "free" rounds might a priori be pressing due to the low total number of rounds, it seems like the built-in countermeasures make such threats impractical.

[4]Except for XHash8, where some inverse monomials are not applied, but this is of no consequence here.

2.4 A Specific Monomial Ordering

In the end, we have two types of variables: the x_j, that correspond to the input, and the y_j^i, that are grouped in layers corresponding to the round at which they are introduced. We order them using a weighted grevlex order, whereby variables x_j have weight 1, and a variable y_j^i (introduced during round i) has weight α^{2i}.

Our system of equations is of the form

$$\begin{cases} 0 &= (y_0^0)^7 - \alpha_0(\vec{x}) \\ & \dots \\ 0 &= (y_{\ell-1}^0)^7 - \alpha_{11}(\vec{x}) \\ 0 &= (y_0^1)^7 - \mu^0\left(\vec{x}, \vec{y^0}\right) \\ & \dots \\ 0 &= (y_{\ell-1}^{N_r})^7 - \mu^0\left(\vec{x}, \vec{y^0}, ..., \vec{y^{N_r-1}}\right) \\ 0 &= \omega_0\left(\vec{x}, \vec{y^0}, ..., \vec{y^{N_r}}\right) \\ & \dots \\ 0 &= \omega_4\left(\vec{x}, \vec{y^0}, ..., \vec{y^{N_r}}\right) \ , \end{cases} \tag{3}$$

where $\ell = 8$ for XHash8 and $\ell = 12$ for XHash12.

By construction, our custom ordering ensures that the leading monomials in all the starting and middle equations are different variables (namely, all the y_j^i). For XHash8, in the case where we only try to force one 0 in the input and in the output, this is exactly a FreeLunch system: the leading monomial in the last equation is of the form $x_0^{49(r+1)}$, meaning that it is yet another variable, and that the whole system is a Gröbner basis out-of-the-box.

Unfortunately, it is not the case for XHash12, or when the number of attacked output blocks (and thus the number of variables in $\vec{x}$) is at least equal to 2. Still, we do have that the leading monomials in all but the final equations are pairwise distinct.

On XHash12. In XHash12, the input variables do not play a specific role since the inverse monomial layers are full. Indeed, we lose the ability to force the input variables to play a specific role in the last round, and thus cannot have the x_j be the only variables in the leading monomials in any equation.

We thus considered alternative strategies to build the system of equations for XHash12, and in particular to build the weights so that they decrease with the number of rounds, thus forcing the x_i to be the leading terms in the first round, the $\vec{y^i}$ to be the leading terms in the second round, etc. However, the inner workings of the round function prevented us from going further: it is "easy" to force y^d to be the leading term of an equation of the form $y^d - P(y,\vec{x})$, but much more complicated to pick a specific variable intervening in the rather dense polynomial P, and have it be the only variable in the leading monomial of such an equation.

In the end, we use the same encoding for XHash12 as we do for XHash8, and in fact we can build both using the same program.

3 Arguing Security Against Gröbner-based Algebraic Attacks

3.1 Towards a Pencil/Paper Argument

Unfortunately, at this stage, we do not know of a way to construct a Gröbner basis for free out of either permutation in the case where several words need to be controlled. However, under a reasonable conjecture, we can derive sufficient bounds. To describe this conjecture, we first need the following definition.

Definition 1 (b-Almost Basal) *We call* b-almost basal (b-AB) *a system of* n *polynomials* P_i *in variables* $(x_0, ..., x_{n-1})$*, where* $b > 0$ *is an integer, when the following properties hold:*

- *Non-degenerate: the ideal spanned by these polynomials is of dimension 0;*
- *Basal part: there exists a monomial order such that the leading term of* P_i*,* $i < b$*, is the monomial* $x_i^{d_i}$ *for some integer* $d_i > 0$*.*

Such a system is represented in Figure 1.

The idea of this notion is to describe a set of multivariate polynomials which is "partially" a Gröbner basis. Indeed, it is sufficient for polynomials to form a Gröbner basis that the leading term in each P_i is of the form $x_i^{d_i}$. In a b-AB system, the first b polynomials have this shape. In fact, an n-AB system is a Gröbner basis.

$$
\text{b-AB system} \left\{ \begin{array}{l} \left.\begin{array}{l} x_0^{d_0} - P'_0(x_0, ..., x_{n-1}) \\ x_1^{d_1} - P'_1(x_0, ..., x_{n-1}) \\ \qquad ... \\ x_{b-1}^{d_{b-1}} - P'_{b-1}(x_0, ..., x_{n-1}) \end{array}\right\} \text{basal part} \\ \\ P_b(x_0, ..., x_{n-1}) \\ P_{b+1}(x_0, ..., x_{n-1}) \\ \qquad ... \\ P_{n-1}(x_0, ..., x_{n-1}) \end{array} \right.
$$

Figure 1: *The structure of an b-AB system, where* $P_i(x) = x_i^{d_i} - P'(x)$*.*

Using this notion, we are ready to state the following conjecture.

Conjecture 1 (Monotonous Ideal Degree Conjecture (MIDC)) *For a b-AB system, the degree* D_I *of the ideal is lower bounded by* $D_I \geq \prod_{i=0}^{b-1} d_i$*.*

The intuition behind this conjecture is simple: the ideal degree corresponds to the number of solutions the system has in the algebraic closure of the underlying field, and this conjecture postulates that this number does not decrease when we take into account the last equations. That is where the term "monotonous" comes from: we assume that the ideal degree increases or stay constant but does not decrease when we consider more equations. To put it differently, this conjecture posits that the actual

Gröbner basis of the system contains the basal part, and that the last equations (those not in the basal part) do not decrease the ideal degree.

This conjecture is obviously true when $b = n$: in this case, the system is a Gröbner basis, and the number of monomials "under the staircase" is indeed $\prod_{i=0}^{n-1} d_i$.

According to our experiments, for a simplified XHash, this conjecture holds. In fact, in our experiments, we do have that $D_I = \prod_{i=0}^{n-1} d_i$, even though the last polynomials have identical leading terms. In that case the last equations participate as if they had different leading terms. Still, we deem it safer at this stage to only take the basal part into account when making a security claim.

On the Security of XHash. Applying our attack strategy against one word in XHash8, i.e. solving a CICO instance with one 0 in the input and one in the output, we simply obtain a FreeLunch [BBL+24] system. It is a particular case of an n-AB system.

Under the MIDA, the ideal degree corresponding to an attack on w words is lower bounded by the complexity of attacking 1 word. Indeed, when $w > 1$, the first equations still form a basal part of the polynomial system, and only the last $w - 1$ equations are not basal. As a consequence, we get a bound on D_I which corresponds to its value in the $w = 1$ case, namely $D_I \geq d^{\ell N_r}$.

For XHash8, even ignoring the presence of the first equation with leading term x_0 among the final equations, we get that $D_I \geq d^{3\times 8} \approx 2^{67}$. Adding the equation in x_0, which also satisfies the criteria to be part of the basal part, we get $D_I \geq d^{3\times 8} \times d^{3\times 2} \approx 2^{84}$. As a consequence, in order for Gröbner-based algebraic attacks to be a threat against XHash8, we would need two things:

1. that the bound for D_I is tight (experiments indicate that it is not), **and**
2. an FGLM-like algorithm with a complexity strictly sub-quadratic in D_I—in fact, a complexity proportional to $D_I^{1.5}$ would give a complexity around 2^{126} in this case: barely an attack.

For XHash12, we get a bound of $D_I \geq d^{3\times 12} \approx 2^{101}$, which is even higher—and thus safer.

At this stage, we expect neither the bounds to be tight nor such an algorithm to exist. Thus, we claim that XHash8 is safe against Gröbner-based algebraic attack.

3.2 Experimental Results

It is easy to design (and implement) a system of equation corresponding to an attack against a greatly simplified XHash8, differing from the real one in the following ways.

- Use $x \mapsto x^3$ as a small S-box,
- Use a lower value for p (namely, $p = 2^{16} - 17$) such that $x \mapsto x^3$ is a permutation,
- Decrease the number of branches, and consider different patterns for the application of the cubic roots.
- resize the big S-box to operate on $\mathbb{F}_p^2$ rather than $\mathbb{F}_p^3$; and allow its replacement by a simple (full) layer of branch-wide monomials (to better understand the impact of big S-boxes).

The attached `Python` script does just that. We call ℓ the number of cubic roots applied in parallel in each round (so that $\ell = 8$ for XHash8 for instance), and w the number both of x_i and of final equations.[5]

[5] We need these to be equal to maintain an ideal of dimension 0.

Confirming the MIDC. Running experiments on a small number of rounds, we have found that the Monotonous Ideal Degree Conjecture held. In fact, a stronger result seems to hold in the case of XHash8: In this case, the ideal degree seems equal to

$$\underbrace{d^{\ell r}}_{y_i} \times \underbrace{d^{k \times 2r}}_{x_j},$$

which is $d^{(k-1)2r}$ times bigger than the bound given by the conjecture.

On the Impact of T. The much denser T operation (operating over $\mathbb{F}_p^3$) does not influence the degree of the ideal. In that sense, it a priori does not provide more resistance against algebraic attacks than a simpler layer of monomials applied on each branch.

However, in practice, it does imply some important properties. First, its greater density means that all the polynomials have an observably higher Hamming weight—a property which gets stronger as the exponent increases. This means that the equations are hard to generate and to manipulate, and that targeting the (elimination of) specific terms is likely to be much more complicated. More importantly, making the polynomials denser is important to prevent algorithm targetting sparse systems to become applicable.

Furthermore, by operating on several words at once, it prevents an attacker from singling out any of them. Being able to do this is important to select which variables to pick as the leading monomials, or to find heuristics to bypass rounds for free. As a consequence, we consider that these layers offer a significant security increase, albeit not one that is directly visible in the ideal degree.

Comparing XHash8 and XHash12. Unsurprisingly, computing the actual value of D_I takes a lot longer for a simplified XHash with a full layer of cubic roots. Intuitively, this makes sense: in this case, the polynomials obtained are further from being a Gröbner basis, and thus SAGE needs to work harder to obtain one in order to compute this quantity.

References

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Transactions on Symmetric Cryptology*, 2020(3):1–45, 2020.

[AKM23] Tomer Ashur, Al Kindi, and Mohammad Mahzoun. Xhash8 and xhash12: Efficient stark-friendly hash functions. Cryptology ePrint Archive, Paper 2023/1045, 2023. https://eprint.iacr.org/2023/1045.

[BBC+23] Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems. New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and jive compression mode. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 507–539. Springer, 2023.

[BBL+24] Augustin Bariant, Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum. The algebraic freelunch efficient gröbner basis attacks against arithmetization-oriented primitives. Cryptology ePrint Archive, Paper 2024/347, 2024. https://eprint.iacr.org/2024/347.

[BBLP22] Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, and Léo Perrin. Algebraic attacks against some arithmetization-oriented primitives. *IACR Transactions on Symmetric Cryptology*, 2022(3):73–101, 2022.

[BCP23] Aurelien Boeuf, Anne Canteaut, and Léo Perrin. Propagation of subspaces in primitives with monomial sboxes: Applications to rescue and variants of the AES. *IACR Transactions on Symmetric Cryptology*, 2023(4):270–298, Dec. 2023.

[BPW06] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. A zero-dimensional Gröbner basis for AES-128. In Matthew J. B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 78–88. Springer, Heidelberg, March 2006.

[Fau99] Jean-Charles Faugere. A new efficient algorithm for computing gröbner bases (f4). *Journal of pure and applied algebra*, 139(1-3):61–88, 1999.

[Fau02] Jean Charles Faugere. A new efficient algorithm for computing gröbner bases without reduction to zero (f 5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, 2002.

[FGLM93] Jean-Charles Faugere, Patrizia Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

[GHR+23] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. Horst meets fluid-spn: Griffin for zero-knowledge applications. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 573–606. Springer, 2023.

[GJMG11] Bertoni Guido, Daemen Joan, P Michaël, and VA Gilles. Cryptographic sponge functions, 2011.

[Hir16] Shoichi Hirose. Sequential hashing with minimum padding. NIST Workshop on Lightweight Cryptography 2016, 2016.

[RST23] Arnab Roy, Matthias Johann Steiner, and Stefano Trevisani. Arion: Arithmetization-oriented permutation and hashing from generalized triangular dynamical systems, 2023.

[Ste24a] Matthias Johann Steiner. Zero-dimensional gröbner bases for rescue-xlix. Cryptology ePrint Archive, Paper 2024/468, 2024. https://eprint.iacr.org/2024/468.

[Ste24b] Matthias Johann Steiner. A zero-dimensional gröbner basis for poseidon. Cryptology ePrint Archive, Paper 2024/310, 2024. https://eprint.iacr.org/2024/310.