Interpolation Cryptanalysis of Unbalanced Feistel Networks with Low Degree Round Functions

Arnab Roy[1], Elena Andreeva[1], and Jan Ferdinand Sauer[2]

[1] Alpen-Adria Universität Klagenfurt, Austria
elena.andreeva@aau.at, arnab.roy@aau.at
[2] KU Leuven, Belgium and KIT, Germany
ferdinand.sauer@posteo.de

Abstract. In recent years a new type of block ciphers and hash functions over a (large) field, such as MiMC and GMiMC, have been designed. Their security, particularly over a prime field, is mainly determined by algebraic cryptanalysis techniques, such as Gröbner basis and interpolation attacks. In SAC 2019 Li and Preneel presented low memory interpolation attack against the MiMC and Feistel-MiMC designs.

In this work we answer the open question posed in their work and show that low memory interpolation attacks can be extended to unbalanced Feistel networks (UFN) with low degree functions, and in particular to the GMiMC design. Our attack applies to UFNs with expanding and contracting round functions keyed either via identical (univariate) or distinct round keys (multivariate). Since interpolation attacks do not necessarily yield the best possible attacks over a binary extension field, we focus our analysis on prime fields $\mathbb{F}_p$.

Our next contribution is to develop an improved technique for a more efficient key recovery against UFNs with expanding round function. We show that the final key recovery step can be reduced not only to the gcd but also to the root finding problem. Despite its higher theoretical complexity, we show that our approach has a particularly interesting application on Sponge hash functions based on UFNs, such as GMiMCHash.

We illustrate for the first time how our root finding technique can be used to find *collision*, *second preimage* and *preimage* attacks on members of the GMiMCHash family. In addition, we support our theoretical analysis with small-scale experimental results.

1 Introduction

In recent years we have seen the advent of novel symmetric cryptographic primitives that aim to facilitate efficiency optimizations for higher level Multi-Party Computation (MPC), Zero-Knowledge (ZK) proofs, or Fully Homomorphic Encryption (FHE) protocols. Examples of such primitives are the LowMC [5] block cipher, MiMC [4], GMiMC [2,3], Starkard and Poseidon [12], MARVELlous [6,7] families of primitives, Kreyvium [10], FLIP [17], and Rasta [11] stream ciphers.

The main design goals of these novel primitives are to reduce the number of multiplications in the (arithmetic) circuit and/or to reduce the multiplicative depth of the circuit corresponding to the underlying block cipher or hash function. These objectives often mean that the primitives must be designed over one (possibly large) finite field and more generally, require thorough rethinking of the way traditional block ciphers (and hash functions) are designed and cryptanalyzed. A recent effort in that direction was the public STARK-Friendly hash challenge [18] which aimed at the evaluation and developing better understanding of the security of the ZK-friendly hash function families Feistel-MiMC, GMiMC, Starkard and Poseidon, and MARVELlous.

The MiMC and Feistel-MiMC (over $\mathbb{F}_p$) family of block ciphers and hash functions are the first ZKP-friendly dedicated symmetric designs. The MiMC block cipher introduced a novel approach towards designing an ZK/MPC-friendly primitive. Unlike previous existing designs, MiMC iterates a low-degree function sufficiently many times to achieve the target security. Both Feistel-MiMC and GMiMC rely on the Feistel iterative structure which builds a permutation from a low degree polynomial round function(s).

The first third party cryptanalysis on the MiMC construction has been conducted by Li and Preneel [15]. Their main idea of retrieving a key dependent coefficient of interpolating polynomial stems from [20]. Li and Preneel apply algebraic cryptanalysis based on Lagrange interpolation with low memory complexity against MiMC. In essence, their attack boils down to finding a key recovery "shortcut" by reconstructing and evaluating the second highest coefficient in the cipher polynomial representation. In their work the authors left the *open question* whether a similar attack approach applies to the GMiMC design. The main reason behind this question we speculate is – it is not obvious how to extend the algebraic analysis in [15] to generalized Feistel networks. Our technical contribution in this article starts with the analytical results that solve this problem.

GMiMC or Generalized MiMC family is an extension of the Feistel-MiMC family and uses generalized Feistel structures with $t > 2$ branches. GMiMC is proposed for *both* balanced and unbalanced Feistel networks. In particular, two unbalanced Feistel networks (UFN) are investigated: with expanding round function (ERF); and with contracting round function (CRF). For both ZKSNARK and MPC applications, the UFN with ERF was found to be more efficient compared to balanced Feistel networks. The GMiMC designers also use a fixed-key GMiMC permutation in a Sponge hashing mode to construct the hash function GMiMCHash. Since the general design principle in GMiMC family relies on a Feistel network with a low degree round function, in this work we focus on the (interpolation) cryptanalysis of UFN with low degree round function over $\mathbb{F}_p$.

1.1 Related work

The MiMC and GMiMC constructions use an APN function to instantiate the round function in the Feistel network. Traditional differential cryptanalysis techniques are ineffective against those designs [2,4]. The design rationale and cryptanalysis of these constructions thus, heavily relies on the *algebraic analysis* of the keyed permutation. Known algebraic cryptanalysis is mainly based on Gröbner basis, interpolation, greatest common divisor (GCD), and higher-order differential analysis. We point out that higher-order differential analysis exploits simply the degree of a keyed function over a field.

In CRYPTO 2020, a higher order differential cryptanalysis against the full (round) GMiMC permutation was proposed [8]. More specifically, the authors showed a zero-sum distinguisher against the GMiMC fixed-key permutation. Yet, they do not provide a collision on the full GMiMCHash using this distinguisher, and to the best of our knowledge no result on finding collision using such distinguisher exists to date. A collision on the reduced round GMiMCHash was given via (algebraic) differential analysis of UFNs. Compared to this our analysis is based on purely algebraic techniques such as polynomial interpolation, GCD computation and root finding.

Bonnetain showed an attack [9] on the Feistel-MiMC and GMiMC n-bit key block ciphers with complexity $2^{n/2}$. The attack followed due to a key schedule weakness and is comparable with the slide attack.

Our results

In this article we analyze UFN constructions over $\mathbb{F}_p$, and their concrete GMiMC instantiation. More specifically, we focus on the interpolation cryptanalysis of UFNs with low degree round functions. Thereafter, we apply our analysis to the GMiMC block cipher over $\mathbb{F}_p$. We further use these ideas to cryptanalyze GMiMCHash ERF and CRF instances.

- We exhibit a low memory interpolation attack on UFNs with both contracting and expanding round functions. Then, we apply our analysis to GMiMC$_{\mathsf{erf}}$ and GMiMC$_{\mathsf{crf}}$ which are UFNs instantiated with the APN function $x \mapsto x^3$. This answers the open question of Li and Preneel [15] – *how to extend the low memory interpolation attack against GMiMC*. The main idea starts with an extension of their low memory (LM) interpolation analysis. Namely, we first construct the key-dependent coefficient of the interpolation polynomial and then recover the coefficient with constant memory. To extend Li and Preneel's attack to our setting we also need to first fix all but one input to a UFN, such that for both ERF and CRF UFNs we can obtain key-dependent polynomial terms with algebraic degree as low as possible. While the method bears similarity with the main idea of [15], we show that a more in-depth analysis (section 3.2) is required to extend the low memory interpolation attacks for UFNs with $t > 2$ branches compared to the two branch FN used to construct Feistel-MiMC. We present a new attack (in section 5.4) which demonstrates how the multiple output branches in UFN$_{\mathsf{erf}}$ can be combined to improve the complexity of the low memory interpolation attack.
- We show how root finding algorithms for polynomials over finite fields can be used instead of the GCD technique. For the target (MPC and ZKP) applications of GMiMC$_{\mathsf{erf}}$ and GMiMC$_{\mathsf{crf}}$ block ciphers and hash functions this method has roughly the same complexity as the GCD one. This is due to the fact that for these applications the size of the prime field is bounded.

- Another important contribution of this work is that we show for the first time how the interpolation analysis together with the root finding technique can be utilized to attack a *hash function* constructed with UFNs of low degree round function in a Sponge hashing mode, such as the GMiMC$_{\mathsf{erf}}$ and GMiMC$_{\mathsf{crf}}$ hash functions. Using this technique we mount collision, second preimage and preimage attacks (in section 7) against these hash functions.
- As a proof-of-concept we apply our cryptanalysis against a few small-scale instantiations of GMiMC$_{\mathsf{erf}}$ and GMiMC$_{\mathsf{crf}}$ block ciphers and provide the experimental results in tables 2 and 3.

  For second preimage, preimage and for collision attacks against GMiMCHash we provide small-scale experiments, the results of which are presented in tables 5 and 6 and figs. 4 and 5.

2 Preliminaries

$\mathbb{F}_p$ denotes the finite field where p is prime. Let k_i denote the rounds keys, while round constants are denoted by c_i for $i \geqslant 0$. We denote the tuple of all subkeys by $K = (k_0, \ldots, k_{r-1})$. When working with fix but unknown key we denote it by $\bar{k}_i$ or $\bar{K}$. The monic polynomial $f(x) = \sum_{j=0}^{d-1} a_j x^j + x^d$ with degree d is used to define the round function in a UFN. We denote the output of $f(x)$ in round i by σ_i. $(x_0, \ldots, x_{t-1}) \in \mathbb{F}_p^t$ is the plaintext or input to a UFN with t (> 2) branches. $P_j^{(i)}$ denotes the polynomial corresponding to branch j after the i-th round, and $P_j^{(0)} = x_j$ for $0 \leqslant j < t$. For the inputs analyzed in section 3 and because $f(x)$ is monic, we have $P_j^{(i)} = x^{d^y} + q(K)x^{d^y-1} + \ldots$ where y depends on r and t. We call the polynomial $q(K)$ in the key K the "second highest coefficient". We illustrate one-round UFNs with ERF and CRF in fig. 1.

2.1 Background

We will consider UFNs with the above polynomial round function $f(x)$. The round function UFN$_{\mathsf{erf}}$ is defined as $\sigma_i = f(P_0^{(i)} + k_i + c_i)$ where $i \geqslant 0$ and $P_j^{(i)} = x_j$ for $i = 0$, $0 \leqslant j < t$. Each round is viewed as a mapping

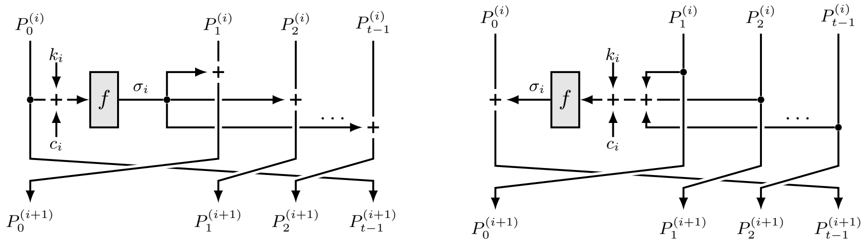
$$(P_0^{(i+1)}, \ldots, P_{t-1}^{(i+1)}) \leftarrow (P_1^{(i)} + \sigma_i, \ldots, P_{t-1}^{(i)} + \sigma_i, P_0^{(i)}) \tag{1}$$

in $\mathbb{F}_p^t$.

The round function of an UFN$_{\mathsf{crf}}$ is defined as $\sigma_i = f(\sum_{j=1}^{t-1} P_j^{(i)} + k_i + c_i)$. Each round of UFN$_{\mathsf{crf}}$ is a mapping defined as

$$(P_0^{(i+1)}, \ldots, P_{t-1}^{(i+1)}) \leftarrow (P_1^{(i)}, \ldots, P_{t-1}^{(i)}, P_0^{(i)} + \sigma_i) \tag{2}$$

We will use the notations UFN$_{\mathsf{erf}}[p, r, t]$ and UFN$_{\mathsf{crf}}[p, r, t]$ to indicate the number of rounds r and number of branches t for UFNs over $\mathbb{F}_p$.



(a) UFN with ERF

(b) UFN with CRF

Fig. 1: One round UFN with ERF(left) and CRF(right)

GMiMC uses the monomial round function $f(x) := x^3$. Typically, GMiMC is defined for primes of size 128 bit or more for their target MPC or ZK applications. The key scheduling in the GMiMC proposal is defined as $k_i = k$ for $i \geqslant 0$ and $k \in \mathbb{F}_p$. However, following the recent attack [9] against GMiMC$_{\mathsf{erf}}$, certain types of GMiMC$_{\mathsf{crf}}$, and Feistel-MiMC, the round key scheduling is updated [16]. The new key schedule defines $k_i := (i + 1)k$ for $i \geqslant 0$. To the best of our knowledge this thwarts the attack of [9]. Using the new key schedule, the round function for GMiMC$_{\mathsf{erf}}$ is defined as $\sigma_i = (P_0^{(i)} + (i + 1)k_i + c_i)^3$, and for GMiMC$_{\mathsf{crf}}$ it is defined as $\sigma_i = (\sum_{j=1}^{t-1} P_j^{(i)} + (i + 1)k_i + c_i)^3$.

We will also use the notations GMiMC$_{\mathsf{erf}}[p, r, t]$ and GMiMC$_{\mathsf{crf}}[p, r, t]$ to indicate the number of rounds and number of branches when necessary or GMiMC.

2.2 Low Memory Interpolation

The interpolation attack [13] on block ciphers was introduced by Jakobsen and Knudsen in 1997. In this attack, the output of a block cipher E_k is viewed as a polynomial in the input of the cipher. The adversary first estimates the degree d of the polynomial and then gathers at least $(d+1)$ pairs of plaintext and ciphertext. This allows reconstruction of E_k without the knowledge of k. The time complexity for reconstructing the polynomial corresponding to E_k is $\mathcal{O}(d \log d)$ [19] and the space complexity is $\mathcal{O}(d)$.

For polynomials of large degree, the amount of required memory can make interpolation impractical. If the involvement of subkeys in a specific coefficient of the polynomial is "well-understood", a partial recovery of the polynomial can be sufficient to recover the key or reconstruct a decryption oracle. In [15], for example, the reconstruction of only a single coefficient allows the recovery of two subkeys.

This lowers the space complexity to $\mathcal{O}(1)$, meaning that the memory constraints are not an issue with this form of low memory interpolation. One such enabling technique is *choosing* the polynomials' evaluation points. Concretely, when interpolating $f : \mathbb{F}_p \to \mathbb{F}_p$, points $(x_j, y_j) = (\alpha^j, f(\alpha^j))$ are used, where $\alpha \in \mathbb{F}_p$ is a primitive element. The y_j are retrieved in an online manner, i. e. "on the fly". This removes the need to store a list of $(d + 1)$ coordinates.

The low memory interpolation algorithm is developed by rearranging the equation for Lagrange interpolation, which is revisited in eq. (3).

$$L(x) = \sum_{j=0}^{d} y_j \prod_{\substack{0 \leq i \leq d \\ i \neq j}} \frac{x - x_i}{x_j - x_i} = \sum_{j=0}^{d} f(\alpha^j) \prod_{\substack{0 \leq i \leq d \\ i \neq j}} \frac{x - \alpha^i}{\alpha^j - \alpha^i} = \sum_{j=0}^{d} b_j x^j \tag{3}$$

Solving for the coefficient b_{d-1}, the authors of [15] arrive at

$$b_{d-1} = \sum_{j=0}^{d} f(\alpha^j) \frac{\beta_j}{\gamma_j}, \quad \gamma_j = \prod_{\substack{0 \leq i \leq d \\ i \neq j}} (\alpha^j - \alpha^i), \quad \beta_j = \alpha^j - \sum_{i=0}^{d} \alpha^i \tag{4}$$

The recursive form of $\gamma_{j+1} = \gamma_j \cdot \alpha^d \cdot {}^{\alpha^j - \alpha^{-1}}/_{\alpha^j - \alpha^d}$ allows its iterative construction. Low memory interpolation can now be achieved by iteratively constructing the summands of eq. (4). Only current values of variables α^j, β_j, γ_j, and the partial result are stored across iterations, resulting in space requirements of $\mathcal{O}(1)$. Despite the improvements on space complexity, interpolation of even one coefficient amounts to time complexity $\mathcal{O}(d \log d)$.

3 Analysis of output polynomials

In this section we will first analyze common properties of the output of Unbalanced Feistel Networks (UFNs) when seen as polynomials of the input and key variable(s). We take output branch j of a $\mathrm{UFN}_{\mathsf{erf}}[p, r, t]$ where the key $K = (k_0, \ldots, k_{r-1})$ values are regarded as indeterminates. Given inputs of the form $(x_0, x_1, \ldots, x_{t-1})$, output branch j can be interpreted as a multivariate polynomial in $\mathbb{F}_p[x_0, \ldots, x_{t-1}, k_0, \ldots, k_{r-1}]$.

Fixing all but one of the input variables to an arbitrary constant will give a polynomial $\mathbb{F}_p[x, k_0, \ldots, k_{r-1}]$ corresponding to any output branch.

3.1 ERF analysis

We analyze the output polynomials corresponding to different branches after r rounds of a $\mathrm{UFN}_{\mathsf{erf}}$. For simplicity we start with the analysis for $\mathrm{UFN}_{\mathsf{erf}}[p, r, 3]$. To give a clear idea of the analysis throughout this section, we progressively generalize it. The first generalization is for the number of rounds r in Proposition 1 and then for t branches in Proposition 2. We simplify the analysis by combining actual round key k'_i and round constant c_i i. e. , $k_i := k'_i + c_i$. Furthermore, we assume $\mathsf{deg}(f) \geqslant 3$.

Proposition 1 *Given an input of the form (b, b, x) to the $\mathrm{UFN}_{\mathsf{erf}}[p, r, 3]$, after $r \geqslant 4$ rounds, the output polynomials $P_0^{(r)}, P_1^{(r)}, P_2^{(r)} \in \mathbb{F}_p[x, k_0, \ldots, k_{r-1}]$ for the 3 branches have the following properties:*

1. *$\mathsf{deg}(P_0^{(r)}) = \mathsf{deg}(P_1^{(r)}) = d^{r-2}$ and $\mathsf{deg}(P_2) = d^{r-3}$*

2. $\mathsf{coeff}(P_0^{(r)}, x^{d^{r-2}}) = \mathsf{coeff}(P_1^{(r)}, x^{d^{r-2}}) = 1$
3. $\mathsf{coeff}(P_0^{(r)}, x^{d^{r-2}-1}) = \mathsf{coeff}(P_1^{(r)}, x^{d^{r-2}-1}) = d^{r-3}(a_{d-1} + d\beta)$
   where $\beta = f(b + k_0) + f(b + f(b + k_0) + k_1) + k_2 = \sigma_0 + \sigma_1 + k_2$.

Proof. The proof is detailed in appendix A.1.

We further generalize the result for t branches in the following proposition:

Proposition 2 *Given an input of the form* $(b, \dots, b, x)$ *to the* $\mathrm{UFN}_{\mathsf{erf}}[p, r, t]$, *let* $r > t \geqslant 3$, *then after* r *rounds, the output polynomials* $P_0^{(r)}, P_1^{(r)}, \dots, P_{t-1}^{(r)} \in \mathbb{F}_p[x, k_0, \dots, k_{r-1}]$ *have the following properties:*

1. $\mathsf{deg}(P_0^{(r)}) = \ldots = \mathsf{deg}(P_{t-2}^{(r)}) = d^{r-(t-1)}$ *and* $\mathsf{deg}(P_{t-1}^{(r)}) = d^{r-t}$
2. $\mathsf{coeff}(P_0^{(r)}, x^{d^{r-(t-1)}}) = \ldots = \mathsf{coeff}(P_{t_2}^{(r)}, x^{d^{r-(t-1)}}) = 1$
3. $\mathsf{coeff}(P_0^{(r)}, x^{d^{r-(t-1)}-1}) = \ldots = \mathsf{coeff}(P_{t-2}^{(r)}, x^{d^{r-(t-1)}-1}) = d^{r-t-1}(a_{d-1} + d\beta)$
   where $\beta = \sum_{i=0}^{t-2} \sigma_i + k_{t-1}$.

Proof. The proof is detailed in appendix A.2.

Corollary 1 *From Proposition 2 we summarize and can further conclude*

1. $\mathsf{deg}(P_t^{(r)}) = \mathsf{deg}(P_1^{(r-1)}) = d^{r-t}$
2. $\mathsf{coeff}(P_t^{(r)}, x^{d^{r-t}-1}) = \mathsf{coeff}(P_1^{(r-1)}, x^{d^{r-t}-1}) = d^{r-t-1}(a_{d-1} + d\beta)$

Corollary 1 gives us the algebraic expression of the coefficient of the second highest degree term in the output polynomial $P_{t-1}^{(r)}$. In the remainder of this article, we will informally refer to this coefficient as the "second highest coefficient." Lastly, we generalize the result for the position of the indeterminate x.

Proposition 3 *Given an input of the form* $(b, \dots, b, x, b, \dots, b)$ *to the* $\mathrm{UFN}_{\mathsf{erf}}[p, r, t]$, *where the position of* x *is* $\ell \in \{0, \dots, t-1\}$, *after* $r > \ell$ *rounds, the output polynomials* $P_0^{(r)}, P_1^{(r)}, \dots, P_{t-1}^{(r)} \in \mathbb{F}_p[x, k_0, \dots, k_{r-1}]$ *have the following properties:*

1. $\mathsf{deg}(P_0^{(r)}) = \ldots = \mathsf{deg}(P_{t-2}^{(r)}) = d^{r-\ell}$ *and* $\mathsf{deg}(P_{t-1}^{(r)}) = d^{r-\ell-1}$
2. $\mathsf{coeff}(P_0^{(r)}, x^{d^{r-\ell}}) = \ldots = \mathsf{coeff}(P_{t-2}^{(r)}, x^{d^{r-\ell}}) = 1$
3. $\mathsf{coeff}(P_0^{(r)}, x^{d^{r-\ell}-1}) = \ldots = \mathsf{coeff}(P_{t-2}^{(r)}, x^{d^{r-\ell}-1}) = d^{r-\ell-1}(a_{d-1} + d\beta)$
   where $\beta = \sum_{i=0}^{\ell-1} \sigma_i + k_\ell$

Proof. Using the same argumentation as in the proof of Proposition 2, we observe that $\mathsf{deg}(\sigma_i) = 0$ for $i < \ell$. In the ℓ-th round, using the same expansion as in the proof of Proposition 2, we have the following expanded form for σ_ℓ:

$$\begin{aligned}\sigma_\ell &= f(x + \sigma_0 + \dots + \sigma_{\ell-1} + k_\ell) \\ &= f(x + \beta) \\ &= x^d + (a_{d-1} + d\beta)x^{d-1} + \dots + a_0\end{aligned}$$

Now we can make an induction over r much in the same way as in Proposition 1.

Corollary 2 *Let σ_i be the output of the round function in round i of a* $\mathrm{UFN}_{\mathsf{erf}}[p,r,t]$ *with input of the form $(b,\dots,b,x,b,\dots,b)$, where indeterminate x is at position ℓ, and $d \geqslant 3$. From the proof of Proposition 3 we have*

$$\mathit{deg}(\sigma_i) = \begin{cases} 0, & 0 \leqslant i < \ell \\ d^{i-\ell+1}, & \ell \leqslant i < r \end{cases}$$

Note that generally, the output polynomials in Proposition 3 are of higher degree than those in Proposition 2, unless $\ell = t-1$, in which case Proposition 2 and Proposition 3 coincide.

3.2 CRF analysis

For the $\mathrm{UFN}_{\mathsf{crf}}[p,r,t]$ we also analyze the degree and coefficients of the highest and second highest term in all the output polynomials, which are described in the following propositions. As in section 3.1, we simplify the analysis by combining actual round key k_i' and round constant c_i to $k_i := k_i' + c_i$. Recall that for a $\mathrm{UFN}_{\mathsf{crf}}$, we use $f(\sum_{i=1}^{t-1} x_i + k_i + c_i)$ as round function in round i on inputs $(x_0,\dots,x_{t-1})$.

Proposition 4 *Given an input of the form $(x,b,\dots,b)$ to the* $\mathrm{UFN}_{\mathsf{crf}}[p,r,t]$ *with $t \geqslant 3$ branches, after $r \geqslant 2$ rounds, the rightmost output polynomial $P_{t-1}^{(r)} \in \mathbb{F}_p[x,k_0,\dots,k_{r-1}]$ has the following properties:*

1. $\mathit{deg}(P_{t-1}^{(r)}) = d^{r-1}$
2. $\mathit{coeff}(P_{t-1}^{(r)}, x^{d^{r-1}}) = 1,$
3. $\mathit{coeff}(P_{t-1}^{(r)}, x^{d^{r-1}-1}) = d^{r-1}(a_{d-1} + d\beta)$
   where $\beta = (t-2)b + f((t-1)b + k_0) + k_1$

Proof. The proof is given in appendix A.3

Corollary 3 *From Proposition 4 we can conclude for $r \geqslant t$*

1. $\mathit{deg}(P_0^{(r)}) = \mathit{deg}(P_{t-1}^{(r-(t-1))}) = d^{r-t}$
2. $\mathit{coeff}(P_0^{(r)}, x^{d^{r-t}-1}) = \mathit{coeff}(P_{t-1}^{(r-(t-1))}, x^{d^{r-t}-1}) = d^{r-t-1}(a_{d-1} + d\beta)$

Corollary 3 gives us the algebraic expression of the coefficient of the second highest degree term in the output polynomial $P_0^{(r)}$. In the remainder of this article, we will informally refer to this coefficient as the "second highest coefficient."

The insight of Proposition 4 allows the algebraic expression of the second highest coefficient in indeterminates k_i, i. e. a polynomial in $\mathbb{F}_p[k_0,k_1]$.

4 Low Memory Interpolation Cryptanalysis of UFNs

Using the results from section 3 we will analyze $\mathrm{UFN}_{\mathsf{erf}}$ and $\mathrm{UFN}_{\mathsf{crf}}$ with uniformly randomly fixed but unknown key $\bar{k} \in \mathbb{F}_p^s$ $(s \geqslant 1)$, resulting in round keys $(\bar{k}_0,\dots,\bar{k}_{r-1})$. When the key values are known the output polynomials developed in sections 3.1 and 3.2, specifically in Corollaries 1 and 3, are elements of $\mathbb{F}_p[x]$ (not of $\mathbb{F}_p[x, k_0, \ldots, k_{r-1}]$).

Since the interpolation of a single coefficient requires low memory, as outlined in section 2.2, we can recover the second highest coefficient to mount a low memory attack on UFN$_{\mathsf{erf}}$ and UFN$_{\mathsf{crf}}$.

4.1 Analysis outline

The general idea of the cryptanalysis can be described in the following steps:

S1. Obtain the algebraic expression of the second highest coefficient $Q(K)$ of the output polynomial corresponding to the branch with the lowest algebraic degree. For UFN$_{\mathsf{erf}}$ and UFN$_{\mathsf{crf}}$ these are rightmost branch and leftmost branch respectively. (Detailed analysis of UFN$_{\mathsf{erf}}$ and UFN$_{\mathsf{crf}}$ are in sections 5.1 and 6 respectively.)

S2. Find value z of second highest coefficient of $E_{\bar{K}}$ of the same branch as in step 1 by applying the low memory interpolation (used in [15]) technique. (Detailed in Section 5.2)

S3. Recover the key by evaluating relation $Q(K) = z$ by solving for K. Some of the key recovering techniques may require multiple equations $Q_i(K) = z_i$. (Sections 5.3 and 6)

Round keys. We explore two scenarios *single key* and *multiple keys*. In single key, $k_i = g(k)$, where g is a *linear*[3] function (and degree one) over $\mathbb{F}_p$ and k can have values from $\mathbb{F}_p$. For single round keys, we use two different techniques: a novel *root finding* technique and the *gcd* technique (previously used by [15]).

In section 5.4, we show that the complexity of the key recovery can be further improved for UFN$_{\mathsf{erf}}$ by combining the output branch polynomials. We call this technique *branch subtraction.* Table 1 gives an overview of the complexities.

In multiple key scenario, the round keys are derived from $k = (k_0, k_1) \in \mathbb{F}_p^2$. In GMiMC, the round key $k_i = k_{i \ (\mathrm{mod}\ 2)}$, where $j \in \{0, \ldots, r-1\}$. The analysis for multiple key scenario is provided in section 5.5.

5 Cryptanalysis of UFN$_{\mathsf{erf}}$

5.1 Algebraic Expression of Second Highest Coefficient

We consider the output polynomial when all but one branch of the inputs have fixed values. By arranging the terms as in Propositions 1 to 3, the polynomial has the form $x^{d^r} + Q(K)x^{d^r - 1} + \cdots + cx^0$, where $Q(K)$ depends on the number of rounds r, the number of branches t, and the position ℓ of indeterminate x in the UFNs input. The coefficient $Q(K)$ is a polynomial that we refer as "second highest coefficient." This coefficient is computable by applying the results from section 3, as described below.

[3] A low degree non-linear function can also be applied. However, the degree of this function will have an effect on the complexity since the time complexity of finding $Q(k)$ depends on the degree.

In a $\mathrm{UFN}_{\mathsf{erf}}[p, r, t]$, the polynomial representing the rightmost output branch has the lowest degree, as shown in Proposition 2. For this polynomial, the coefficient of the second highest degree term has form $Q(K) = d^{r-t-1}(a_{d-1}+d\beta)$, with $\beta = \sum_{i=0}^{t-2} \sigma_i + k_{t-1}$. Algorithm 1 describes the method to obtain the polynomial $Q(K)$, representing the second highest coefficient.

Complexity (of S1). In the single key case, the computation of $Q(K)$ requires multiplications of polynomials over $\mathbb{F}_p$. More specifically, it requires at most $\mathcal{O}(\log D)$ multiplications of two polynomials with degree at most D. The multiplication of two polynomials of degree at most D over $\mathbb{F}_p$ requires $\mathsf{M}(D) := 63.43 D \log D \log\log D + \mathcal{O}(D \log D)$ field operations [22, Thm. 8.23]. To simplify the expressions we define $\mathsf{M}'(D) = \mathcal{O}(\log D)\, \mathsf{M}(D)$. Hence, this step has complexity

$$\mathsf{M}'(d^{t-1}) = \mathcal{O}\big(\log d^{t-1}\big)\, \mathsf{M}(d^{t-1}).$$

Space complexity is $\mathcal{O}\big(d^{t-1}\big)$ since only one polynomial of degree at most d^{t-1} has to be stored at any given time. Note that this space complexity is not due to the interpolation part of the attack.

5.2 Value of Second Highest Coefficient

As outlined at the beginning of section 4, the second step of the analysis consists of recovering the value of the second highest coefficient of the rightmost output polynomial branch of the $\mathrm{UFN}_{\mathsf{erf}}$. For this step we use the low memory interpolation of [15] described in section 2.2.

In general, inputs of form α^j are required, where $\alpha \in \mathbb{F}_p$ is a primitive element, $0 \leqslant j \leqslant D$, and D is the degree of the underlying polynomial that is to be interpolated. In the current analysis this means using inputs of the form $x_j = (b, \ldots, b, \alpha^j)$, in accordance with Proposition 2. The evaluation points y_j for the interpolation are the values of the rightmost output branch.

Complexity (of S2). The time complexity of finding the value of the second highest coefficient using the low memory interpolation method of [15] is $\mathsf{I}_{lm}(D) := \mathcal{O}(D \log(D))$ for polynomials of degree D. Its memory complexity is in $\mathcal{O}(1)$, and data complexity is $D + 1$. For $\mathrm{UFN}_{\mathsf{erf}}$, we have $D = d^{r-t}$, resulting in time

Algorithm 1: Second highest coefficient of rightmost branch in $\mathrm{UFN}_{\mathsf{erf}}[p, r, t]$ on input $(b, \ldots, b, x)$.

```
Input: $r, t, f$, branch constant $b$, round constants $c_0, \ldots, c_{t-1}$
Output: polynomial $Q(K)$ for second highest coefficient of rightmost branch
$s := 0$
for $i \in (0, \ldots, t-2)$ do
    $\sigma_i := f\,(s + b + c_i + k_i)$
    $s := s + \sigma_i$
$\beta := s + k_{t-1} + c_{t-1}$
return $d^{r-t-1}(a_{d-1} + d\beta)$
```

complexity $\mathcal{O}((r-t)d^{r-t}\log d)$. The approach requires $d^{r-t}+1$ pairs of plaintext and ciphertext and uses $\mathcal{O}(1)$ space. Better time and data complexities can be achieved by combining branches, as described in section 5.4.

5.3 Key recovery with a single key

First, we consider the case of single key e.g., $k_i = (i+1)\cdot k$ for $i \geqslant 0$ and k is the master key that can take any value in $\mathbb{F}_p$. We find the polynomial representing the second highest coefficient and the value of the second highest coefficient as described in the previous two sections. For finding the value of the secret key two different techniques can be employed: Finding the gcd, or finding roots.

Finding the GCD. This technique was introduced in [1] and also used in [15] to analyze two-branch Feistel networks. We first select two different constants b, b' for input to the $\mathrm{UFN}_{\mathsf{erf}}$. We obtain two polynomials $Q(k)$ and $Q'(k)$ as described in algorithm 1 using b and b' respectively. Next, we interpolate the values of the second highest coefficient say z, z' with as described in section 5.2, twice: The correct key is found from $\gcd(Q(k)-z, Q'(k)-z')$ with high probability.
Complexity of GCD Finding the gcd of two polynomials of degree at most D over $\mathbb{F}_p$ has time complexity $\mathsf{G}(D) := \mathcal{O}\big(D\log^2 D\big)$ [22, Cor. 11.9]. For $\mathrm{UFN}_{\mathsf{erf}}$s, we have $D = d^{t-1}$. Hence the key recovery using the gcd method has time complexity $\mathcal{O}\big(t^2 d^{(t-1)}\log^2 d\big)$ and space complexity $\mathcal{O}\big(d^{t-1}\big)$.

Finding roots. By construction of $Q(k)$ it satisfies $Q(\bar{k}) - z = 0$, i.e., correct key $\bar{k}$ is a root of above equation $Q(k)-z$. Exhaustively trying the generally very short list of key candidates [14] on one additional pair of plaintext and ciphertext identifies the correct key.
Complexity of root finding Finding all roots without multiplicity of a polynomial with degree D over $\mathbb{F}_p$ has time complexity $\mathsf{R}(D) := \mathcal{O}(\mathsf{M}(D)\log D\log(Dp))$ [22, Cor. 14.16]. Checking at most D key candidates for a polynomial of degree D has time complexity $\mathcal{O}(D)$. For $\mathrm{UFN}_{\mathsf{erf}}$, we have $D = d^{t-1}$. Hence the key recovery using the root finding method has time complexity $\mathcal{O}\big(\mathsf{M}(d^{t-1})\log d^{t-1}\log(d^{t-1}p)\big)$ and data complexity of $\mathcal{O}(1)$.

Finding roots versus GCD. Asymptotically, the complexity of the root finding method is worse than using the gcd method. However, for our target constructions, i.e., GMiMC block ciphers and hash functions that are aimed for practical applications, the complexities are roughly the same, since the field's size $\log p \leqslant 2^8$. More importantly, the root finding method can also be used to find collisions when $\mathrm{UFN}_{\mathsf{erf}}$ (with a fixed key) is used in sponge mode to construct a hash function, as described in section 7.

5.4 Complexity improvements via Branch Subtraction

When analyzing a $\mathrm{UFN}_{\mathsf{erf}}[p, r, t]$ $E_{\bar{K}}$ with $\bar{K} = (\bar{k}_0, \ldots, \bar{k}_{r-1})$, improvements on the complexities discussed above are possible. From Corollary 2 we have $\mathsf{deg}(\sigma_i) = d^{i-\ell-1}$ for $i \geqslant \ell$ for input of the form $(b, \ldots, b, x, b, \ldots, b)$, where $b \in \mathbb{F}_p$ is a constant and indeterminate x is at position ℓ. After round i, by construction of UFN$_{\mathsf{erf}}$, σ_i has been added to all branches except the rightmost one. As has been used extensively in the proofs of Propositions 1 to 3, the degree of the output polynomial of any branch is dominated by the largest σ_i. Thus, somehow removing one or more of the highest σ_i from an output branch reduces the degree of the corresponding polynomial. A lower degree in turn allows interpolation with reduced time and data complexity. Since we're using the low memory technique of [15], space complexity cannot be lowered further.

As an example for this effect, consider the output branches $P_0^{(5)}(x), \dots, P_3^{(5)}(x)$ in fig. 2. We set $P'(x) := P_1^{(5)}(x) - P_0^{(5)}(x) = \sigma_1 - \sigma_2$. While $\mathsf{deg}(P'(x)) = 0$, crucially $\min_i(\mathsf{deg}(P_i^{(5)}(x))) = \mathsf{deg}(P_3^{(5)}(x)) = \mathsf{deg}(\sigma_3) = d$. This elimination of high degree σ_i is the basic idea behind branch subtraction.

We represent the output of a UFN$_{\mathsf{erf}}[p, r, t]$ E_K with input $(x_0, \dots, x_{t-1})$ as a vector $\vec{\sigma}$ using the following matrix notation. Intuitively, the matrix A permutes the inputs like the last operation in any one round of a UFN$_{\mathsf{erf}}$. The matrix B accumulates the necessary σ_i, following the definition of a UFN$_{\mathsf{erf}}$.

$$\vec{\sigma} := A^r \cdot \vec{x} + \underbrace{(B_{r\mathtt{mod}t}|\overbrace{B|\dots|B}^{\lfloor\frac{r}{t}\rfloor \text{ times}})}_{r \text{ columns}} \cdot \vec{\sigma} \tag{5}$$

where

$$A = \begin{pmatrix} — e_2 — \\ — e_3 — \\ \vdots \\ — e_t — \\ — e_1 — \end{pmatrix}, \ \vec{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{t-1} \end{pmatrix}, \ B = \begin{pmatrix} 0 & & \mathbf{1} \\ & 0 & \\ \mathbf{1} & & \ddots \\ & & 0 \end{pmatrix}, \ \vec{\sigma} = \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_{r-1} \end{pmatrix}$$

and $B_{r\mathtt{mod}t}$ are the right $r \ \mathtt{mod} \ t$ columns of B. Summarizing the dimensions, we have $A, B \in \mathbb{F}_p^{t \times t}$, $\vec{x} \in \mathbb{F}_p^t$, and $\vec{\sigma} \in \mathbb{F}_p^r$.

Note that eq. (5) is not recursive. Increasing r to $r+1$ leads to different dimensions in the composite matrix on the right hand side as well as in $\vec{\sigma}$. Note also that the output branches $\vec{\sigma}$ are nonlinear in variable x despite the seemingly linear representation above, since the σ_i are nonlinear for $i \geqslant \ell$.

As an example, we consider a UFN$_{\mathsf{erf}}[p, 5, 4]$ with inputs (b, b, b, x) like in fig. 2. In this instance, we have the following:

$$\vec{\sigma} = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 1\,0\,0\,0 \end{pmatrix}^5 \cdot \begin{pmatrix} b \\ b \\ b \\ x \end{pmatrix} + \begin{pmatrix} 1\,0\,1\,1\,1 \\ 1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1 \\ 0\,1\,1\,1\,0 \end{pmatrix} \cdot \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \end{pmatrix} = \begin{pmatrix} b \\ b \\ x \\ b \end{pmatrix} + \begin{pmatrix} \sigma_0 + \sigma_2 + \sigma_3 + \sigma_4 \\ \sigma_0 + \sigma_1 + \sigma_3 + \sigma_4 \\ \sigma_0 + \sigma_1 + \sigma_2 + \sigma_4 \\ \sigma_1 + \sigma_2 + \sigma_3 \end{pmatrix}$$

Thus, $\vec{\sigma}$ is an alternative representation of $P_0^{(5)}(x), \dots, P_3^{(5)}(x)$, concluding the example.

Given this representation of the output branches $\vec{o}$ of E_K, we can apply some linear algebra in the following way: First, we observe the inverse of matrix B.

$$B^{-1} = \frac{1}{t-1}\begin{pmatrix} 2-t & & \text{\Large 1} & \\ & 2-t & & \\ & & \ddots & \\ \text{\Large 1} & & & 2-t \end{pmatrix}$$

Multiplying the vector of output branches $\vec{o}$ by B^{-1} limits occurrence of any σ_i in any one component of $\vec{o}$ to exactly once. This corresponds to every σ_i occurring on only one "combined output branch".

$$B^{-1}\cdot\vec{o} = B^{-1}\cdot A^r\cdot\vec{x} + (\underbrace{I_{r\mathtt{mod}t}|\overbrace{I_t|\dots|I_t}^{\lfloor\frac{r}{t}\rfloor\ \text{times}}}_{r\ \text{columns}})\cdot\vec{\sigma} \tag{6}$$

where $I_t \in \mathbb{F}_p^{t\times t}$ is the identity matrix and $I_{r\mathtt{mod}t}$ are the right $r \ \mathtt{mod}\ t$ columns of I_t.

From eq. (6) in combination with Corollary 2 we can derive

$$\begin{aligned} &\mathsf{deg}(\mathsf{first\_component}(B^{-1}\cdot\vec{o})) \\ &= \mathsf{deg}(\sigma_{r-t} + \sigma_{r-2t} + \cdots + B^{-1}A^r\vec{x}) = \mathsf{deg}(\sigma_{r-t}) \\ &= d^{r-2t+2} \end{aligned} \tag{7}$$

for $\ell = t-1$.

Complexity improvements Using the polynomial $\mathsf{first\_component}(B^{-1}\cdot\vec{o})$ of eq. (7) instead of the rightmost branch in the analysis of sections 5.1 to 5.3 lowers the complexities involved. Step 1 and 3 are unaffected by branch subtraction since the complexities do not depend on the number of rounds r. For S2, the computational complexity is $\mathsf{I}_{lm}(d^{r-2t+2})$ as opposed to $\mathsf{I}_{lm}(d^{r-t})$.

Data complexity with branch subtraction is $d^{r-2t+2}+1$ as opposed to $d^{r-t}+1$ without. Space complexity stays $\mathcal{O}(1)$ since the same low memory algorithm for recovery is being used. A summary of all the complexities with and without branch subtraction can be found in section 5.6.

In order to achieve these improvements, B^{-1} needs to be applied to the output branches $\vec{o}$. Since we only need the first component of vector $B^{-1}\cdot\vec{o}$, we can limit ourselves to one product between two vectors of length r, where each of the components has degree at most d^{r-t+1}. The time complexity for r many multiplications of two polynomials of degree at most d^{r-t+1} is $r\mathsf{M}(d^{r-t+1})$.

5.5 Key recovery with multiple keys

Our target case for multiple keys always reduces to keys $(k_0, k_1) \in \mathbb{F}_p^2$, i. e. keys k_i for $i \geqslant 2$ do not influence the analysis, we interpret round keys k_i as derived from k_0 and k_1.

We consider the round key scheduling $k_i = (i+1) \times k_{i \pmod 2}$ for $i \geqslant 0$. As opposed to the variant where the same key is added in every round, the methods from section 5.3 building on Proposition 2 are not directly applicable. Instead, we use the results of Proposition 3. Furthermore, we use multiple instances of the equation $Q_i(K) = z_i$ for different constants b_i for $i \in \{0,1,2\}$. This is an adaptation of the approach used in [15] where the authors analyzed balanced Feistel networks.

In Proposition 3, let $\ell = 1$, which corresponds to inputs of the form $(b_i, x, b_i, \dots, b_i)$. Then, the second highest coefficient of the rightmost branch of E_K is of the form $Q_i(K) = d^{r-2}(a_{d-1} + d\beta_i)$ where $\beta_i = \sigma_0 + k_1 = f(b_i + k_0) + k_1$. Thus, we have $Q_i(K) \in \mathbb{F}_p[k_0, k_1] \subsetneq \mathbb{F}_p[K]$.

We combine the three equations $Q_i(K) = z_i$ in the following manner: By solving for β_i we have

$$\begin{aligned} \beta_0 &= f\left(b_0 + k_0\right) + k_1 = \frac{z_0}{d^{r-1}} - \frac{a_{d-1}}{d} \\ \beta_1 &= f\left(b_1 + k_0\right) + k_1 = \frac{z_1}{d^{r-1}} - \frac{a_{d-1}}{d} \\ \beta_2 &= f\left(b_2 + k_0\right) + k_1 = \frac{z_2}{d^{r-1}} - \frac{a_{d-1}}{d} \end{aligned} \tag{8}$$

For $0 \leqslant i, j \leqslant 2$ we get, through subtracting and rearranging,

$$\Delta_{(i,j)} := f(b_i + k_0) - f(b_j + k_0) - \frac{z_i - z_j}{d^{r-1}} = 0 \tag{9}$$

As in section 5.3, it holds by the factor theorem that $(k_0 - \bar{k}_0)$ is a factor of $\Delta_{(i,j)}$ due to the construction of $Q_i(K)$. $\bar{k}_0$ can be found by computing the gcd $\left(\Delta_{(0,1)}, \Delta_{(0,2)}\right)$. Substituting k_0 with $\bar{k}_0$ in any of eq. (8) yields $\bar{k}_1$.

$$\bar{k}_1 = \frac{z_0}{d^{r-1}} - \frac{a_{d-1}}{d} - f\left(b_0 + \bar{k}_0\right) \tag{10}$$

Complexity (of S1 and S2) Computing the algebraic form of the second highest coefficient can be done in constant time and space. For the complexities of recovering the value of the second highest coefficient, we refer to section 5.2, restating the computational complexity of $\mathsf{I}_{lm}(d^{r-1})$ here. Data complexity is $3d^{r-1} + 3$.

Complexity (of S3) Computing the gcd of polynomials of degree D has computational complexity $\mathsf{G}(D) = \mathcal{O}\big(D \log^2 D\big)$, as discussed in section 5.3. Since, $D = d$, the complexity for recovering $\bar{k}_0$ is $\mathsf{G}(d)$. Once, we recover k_0, the k_1 can be found by using one of the eq. (8).

5.6 Summary of Complexities

In the sections above, a few approaches for key recovery are proposed. In the case of a single key $\bar{k}$, i. e. $\bar{K} = (\bar{k}, \dots, \bar{k})$, we pointed out a novel method using root finding and applied an existing method using the gcd. The time and data complexities of the different approaches are summarized in table 1. Furthermore,

Primitive	strategy	time	data	section
ERF	root	$\mathsf{M}'(d^{t-1}) + \mathsf{I}_{lm}(d^{r-t}) + \mathsf{R}(d^{t-1})$	$\mathcal{O}\big(d^{r-t}\big)$	5.3
	gcd	$\mathsf{M}'(d^{t-1}) + \mathsf{I}_{lm}(d^{r-t}) + \mathsf{G}(d^{t-1})$	$\mathcal{O}\big(d^{r-t}\big)$	5.3
	root (bs)	$\mathsf{M}'(d^{t-1}) + \mathsf{I}_{lm}(d^{r-2t+2}) + \mathsf{R}(d^{t-1})$	$\mathcal{O}\big(d^{r-2t+2}\big)$	5.3, 5.4
	gcd (bs)	$r\mathsf{M}(d^{r-t+1}) + \mathsf{I}_{lm}(d^{r-2t+2}) + \mathsf{G}(d^{t-1})$	$\mathcal{O}\big(d^{r-2t+2}\big)$	5.3, 5.4
	multikey	$\mathsf{I}_{lm}(d^{r-1}) + \mathsf{G}(d)$	$\mathcal{O}\big(d^{r-1}\big)$	5.5
CRF	root	$\mathsf{I}_{lm}(d^{r-t}) + R(d)$	$\mathcal{O}\big(d^{r-t}\big)$	6
	gcd	$\mathsf{I}_{lm}(d^{r-t}) + G(d)$	$\mathcal{O}\big(d^{r-t}\big)$	6
	multikey	$\mathsf{I}_{lm}(d^{r-1}) + \mathsf{G}(d)$	$\mathcal{O}\big(d^{r-t}\big)$	6

Table 1: Complexities of low memory interpolation cryptanalysis for $\mathrm{UFN}_{\mathsf{erf}}[p, r, t]$ for $r > 2t$ and $\mathrm{UFN}_{\mathsf{crf}}[p, r, t]$ for $r > t$. The branch subtraction technique of section 5.4 is denoted by "bs".$\mathsf{M}, \mathsf{M}', \mathsf{I}_{lm}, \mathsf{G}$ and R are defined in section 5.2 and section 5.1.

the sections describing the approaches are pointed out. Since the algebraic degree of the round function d is assumed to be small, space complexities are omitted in this overview. In most cases, the interpolation step dominates the computational as well as data complexity.

5.7 Experimental Verification

We have validated our analysis by running small scale experiments. The UFN instances use randomized key, round constants, and coefficients for the round function. Since our analysis considers monic round functions, the highest coefficient of the round function is always 1. The fixed parameters of our experiments are $q = 99999989, r = 17, t = 4$. The round function is of degree 3. We use both proposed methods of key recovery, namely root finding and the gcd method, and apply the branch subtraction technique of section 5.4. Given above parameters, the degree of the combined output polynomial for $\mathrm{UFN}_{\mathsf{erf}}$ is 3^{11}.
It is interesting to observe the average number of roots: Although up to $3^{4-1}+1 = 10$ roots could occur in theory, our experiments show that in practice, this number is significantly lower, with an average of only 1.89 roots.

	root	gcd
algebraic coefficient	0.08950	0.10453
coefficient value	1 468.50751	3 132.21874
key recovery	0.80580	0.03565
total	1 469.40283	3 132.35892

Table 2: Observed average running times in milliseconds for key recovery of $\mathrm{UFN}_{\mathsf{erf}}[p, 17, 4]$ using root finding and the gcd method. The degree of the interpolated polynomial was 3^{11}. $(n = 100)$

Our experiments are implemented in python using sagemath [21]. All random values were generated using python's built in "random" module. Measurements were taken on a machine with a standard Intel Core i5-6300U CPU and 7.22 GiB of RAM. Each experiment was run $n = 100$ times. A summary of the observed average running times can be found in table 2. We note that the discrepancy of about factor 2 between the two applied methods comes from the fact that the gcd method requires interpolating two polynomials. For the root finding method, only one polynomial is required.

6 Cryptanalysis of $\mathrm{UFN}_{\mathsf{crf}}$

We now analyze UFNs in the CRF variant according to the steps outlined in section 4. Since the analysis is extremely similar to the ERF variant, covered in section section 5, we only point out significant differences. Notably, recovering the value of the second highest coefficient as well as key recovery with only one round key $\bar{k}$, i. e. $\bar{K} = (\bar{k}, \ldots, \bar{k})$ are not reiterated herein.

Algebraic Expression of Second Highest Coefficient. In a $\mathrm{UFN}_{\mathsf{crf}}[p, r, t]$, the polynomial representing the leftmost output branch has the lowest degree, as shown in Proposition 4. For this branch, the second highest coefficient $Q(K)$ has form $d^{r-t-1}(a_{d-1} + d\beta)$, with $\beta = (t-2)b + f((t-1)b + k_0) + k_1$, as shown in Corollary 3. This coefficient is simpler than that of a $\mathrm{UFN}_{\mathsf{erf}}$, as it depends only on k_0 and k_1. Consequently, computing $Q(K)$ is simpler, as described in algorithm 2.

Algorithm 2: Second highest coefficient of leftmost branch in $\mathrm{UFN}_{\mathsf{crf}}[p, r, t]$ on input $(x, b, \ldots, b)$.

```
Input: $r, t, f$, branch constant $b$, round constants $c_0, c_1$
Output: polynomial $q(K)$ for second highest coefficient of leftmost branch
$\beta := f\left((t-1)b + k_0 + c_0\right) + (t-2)b + k_1 + c_1$
return $d^{r-t-1}(a_{d-1} + d\beta)$
```

Complexity (of S1). Calculating the algebraic form of the second highest coefficient requires only addition and multiplication of scalars. Thus, the complexity is $\mathcal{O}(1)$.

Value of Second Highest Coefficient. We recover the second highest coefficient of the *leftmost* branch for $\mathrm{UFN}_{\mathsf{crf}}$. Consequently, evaluation points y_j for the interpolation are the values of the leftmost output branch. Inputs of the form $x_j = (\alpha^j, b, \ldots, b)$ are used for the low memory interpolation, where $\alpha \in \mathbb{F}_p$ is a primitive element as before. These changes allow application of Corollary 3.
Complexity (of S2). The complexities do not change from those of $\mathrm{UFN}_{\mathsf{erf}}$ in section 5.2, i.e., it remains $\mathsf{I}_{lm}(d^{r-t})$.

	root	gcd
algebraic coefficient	0.03036	0.04744
coefficient value	11 832.86184	23 482.13424
key recovery	0.48999	0.02876
total	11 833.38219	23 482.21044

Table 3: Observed average running times in milliseconds for key recovery of $\mathrm{UFN}_{\mathsf{crf}}[p, 17, 4]$ using root finding and the gcd method. The degree of the interpolated polynomial was 3^{13}. ($n = 100$)

Complexity (of S3). This is the final key recovery step. For the case of single key, the complexity of this step using GCD technique is $\mathsf{G}(d)$ and using root finding technique is $\mathsf{R}(d)$.

Key recovery in the general case. We now consider $\mathrm{UFN}_{\mathsf{crf}}$ $E_{\bar{K}}$ with general $\bar{K}$, starting with the bivariate case. That is, $\bar{K} = (\bar{k}_0, \ldots, \bar{k}_{r-1})$ where $\bar{k}_i$ is derived from $\bar{k}_0$ and $\bar{k}_1$. Like in section 5.5, we consider the round key scheduling $k_i = (i + 1) \times k_{i \pmod 2}$ for $i \geqslant 0$. The algebraic form of the second highest coefficient is $Q_i(K) = d^{r-1}(a_{d-1} + d\beta_i)$ where $\beta_i = (t-2)b_i + f((t-1)b_i + k_0) + k_1$, as shown in Proposition 4.

Combining the three equations $Q_i(K) = z_i$ works the same way as in section 5.5. Due to the different form of β_i, the equations change slightly: By solving for β_i we have

$$\beta_i = (t-2)b_i + f((t-1)b_i + k_0) + k_1 = \frac{z_i}{d^r} - \frac{a_{d-1}}{d} \tag{11}$$

For $2 \geqslant i, j \geqslant 0$ we get, through subtracting and rearranging,

$$\varDelta_{(i,j)} := f((t-1)b_i + k_0) - f((t-1)b_j + k_0) + (t-2)(b_i - b_j) + \frac{z_i - z_j}{d^r} = 0 \tag{12}$$

Now, as in section 5.3, we recover $\bar{k}_0$ by finding gcd $\left(\varDelta_{(0,1)}, \varDelta_{(0,2)}\right)$. Then $\bar{k}_1$ can be found using one of the eq. (11).

Complexity (of S1-S3). Although the form of $q(k)$ and β are slightly different from those in $\mathrm{UFN}_{\mathsf{erf}}$, the steps are fundamentally the same as for the ERF variant. Thus, the complexities don't differ from those in section 5.5. We summarize the complexities of the different key recovery approaches for $\mathrm{UFN}_{\mathsf{crf}}$ in table 1.

Experimental Verification Just as for $\mathrm{UFN}_{\mathsf{erf}}$ in section 5.7, we performed small scale experiments for $\mathrm{UFN}_{\mathsf{crf}}$ with the same number of rounds $r = 17$ and branches $t = 4$. Because the branch subtraction technique of section 5.4 does not apply to $\mathrm{UFN}_{\mathsf{crf}}$, the degree of the polynomial that is to be interpolated is 3^{13}. Running times of the experiments can be found in table 3. We note that for the root finding technique, the average length of the key candidate list was 2.1, extremely similar to those of the experiments on $\mathrm{UFN}_{\mathsf{erf}}$ of section 5.7.

7 Cryptanalysis of UFN based Sponge hash

Either of the UFNs may be used in a Sponge mode to construct a hash function over $\mathbb{F}_p$. The permutation thereof is instantiated with a fixed key UFN. An example of such hash function is recently proposed GMiMCHash [2]. Here we will describe how the root finding technique can be used to find collision in such hash function. We will assume that the rate in Sponge mode is $r = \log_2 p$ (bits) and the hash value also has the same size. An input message to the hash function $\in \mathbb{F}_p^s$ for $s \geqslant 1$ and consists of message block that are elements of $\mathbb{F}_p$.

Preimage and second preimage attack Let us consider a message $M' = (m_0', m_1') \in \mathbb{F}_p^2$ of size $2r$ bits. It is clear that the UFN must have branch size r bits. Suppose, f denotes the permutation instantiated with a fixed key UFN and the rightmost output branch corresponds to the rate in Sponge mode. We find a second preimage M in the following way

1. Choose arbitrary message block $m_0 \in \mathbb{F}_p$. Suppose, h_t $(\in \mathbb{F}_p)$ denotes the first r bits (and the rightmost output branch) of $f(m_0)$ and h_i for $i = 2, \ldots t$ denote the outputs corresponding to the $t-1$ branches of the UFN. Let h be the hash value of the message $M' = (m_0', m_1')$.
2. Compute the polynomial $P(x)$ corresponding to the first output branch of the UFN for the input $h_1, h_2, \ldots, h_t + x$. Note that $P(x)$ is the polynomial corresponding to the hash value of (m_0, x).
3. Find the roots of $P(x) - h$.

Note that any root of $P(x) - h$ gives a second preimage attack.

Instead of choosing h as hash value of a message if we choose it arbitrarily then the above attack gives a *preimage* attack on the hash function.

Complexity. The complexity of finding all roots (without multiplicities) is of a polynomial of degree D over $\mathbb{F}_p$ is $\mathsf{R}(D)$. For $\text{UFN}_{\mathsf{erf}}$ the degree of the polynomial $P(x)$ after r rounds is d^{r-t}. Hence, the the complexity of the root finding step is $\mathsf{R}(d^{r-t})$

Collision attack Choose two message blocks $m_0, m_0' \in \mathbb{F}_p$.Then, a collision attack on the hash function is described as following

1. Compute the polynomial $P(x)$ representing the hash value of a message of the form (m_0, x) and $P'(x)$ corresponding to the hash value of the message (m_0', x).
2. Compute the roots of $Q(x) := P(x) - P'(x)$.

The complexity of the collision attack is same as the complexity of the preimage or second preimage attacks. The degree of the polynomial Q is d^{r-t-1}.

Increasing the rate If the rate is increased to $2r$ then the attacks apply similarly. For preimage and second preimage attack we choose the message $M = ((m_{00}', m_{01}'), (m_{10}', m_{11}'))$ where each $m_{ij} \in \mathbb{F}_p$. The polynomial is constructed for $((m_{00}, m_{01}), (m_{10}, x))$. The collision attack also applies analogously as for rate r. The hash output is in $\mathbb{F}_p$. The complexities remains the same since the degree of the does not change.

We point out that the number of rounds for the $\text{UFN}_{\mathsf{erf}}$ given in the GMiMC proposal is not explicitly justified by analyzing the security of the hash function.

7.1 Experimental verification

We evaluate our results from section 7 by running small scale experiments. We regarded $\text{GMiMC}_{\mathsf{erf}}[p, r, t]$ permutation with $p = 99999989$, $3 \leqslant r \leqslant 7$, $3 \leqslant t \leqslant 6$ with $k = 0$ to instantiate the hash function for two different sets of experiments: Finding second preimages and finding collisions. The round constants were randomly chosen and fixed. For each combination of (r, t) in the given intervals, 100 experiments were performed. The messages were re-randomized for every experiment. The experiments were implemented in python using sagemath. To generate the random values, pythons "random" module was used. All the measurements were taken on a machine with a standard Intel Core i5-6300U CPU and 7.22 GiB of RAM.

For the experiments on second preimages, across all 2000 experiments, we observed a total of 751 iterations where no second preimage was found. We consider these experiments to have failed. This puts the estimated success probability of finding at least one preimage to 62.5%. Of secondary interest is the average number of second preimages found given that the attack was successful, i. e. at least one second preimage was found. Over all the 1249 successful experiments, an average of 1.55 second preimages were observed per experiment.

For the experiments on collisions, no collision could be found in 712 of the 2000 experiments. We consider these experiments to have failed. The success probability is thus 64.4%. Of secondary interest is the average number of collisions found in the successful experiments, i. e. at least one collision was found. In 1288 successful experiments, an average of 1.57 collisions were observed per experiment. The failure rates of 37.6% and 35.6% respectively are supported by the fact that for our parameters (p and degree of the underlying polynomial), $\approx 36.8\%$ of the polynomials do not have a root [14]. We elaborate on this in appendix D.

Tables 5 and 6 in appendix C summarize running times of our experiments on second preimages and collisions. In fig. 4 in appendix C, the number of second preimages found in our experiments is plotted. Similarly, fig. 5 in appendix C visualizes the number of collisions found.

8 Attacks on Reduced Round GMiMC

The GMiMC family has two members that are based on UFN: $\text{GMiMC}_{\mathsf{erf}}$ and $\text{GMiMC}_{\mathsf{crf}}$. The round function used in both variants is $f(x) = x^3$, i.e.,$d = 3$. Due to target applications we can also assume that field size is bounded e.g. $\log_2 p \leqslant 256$. For some specific sizes of the field and number of branches in the UFNs, we show the number of rounds of GMiMC block cipher and hash functions that can be attacked in table 4. Our analysis of the GMiMCHash instances does not contradict their security claims in the GMiMC proposal [2] (in terms of number of secure rounds).

Recently, in [8] a collision attack on GMiMCHash-256 was proposed for a field $\mathbb{F}_q$ with $q = 2^{125} + 266 \times 2^{64} + 1$ and $t = 14$. This means the hash output is $\mathbb{F}_q^2$. However, our described method is designed for a hash output in $\mathbb{F}_p$. Since the choice of hash outputs over a field in the two cases are incomparable, a direct

Primitive	type	$\log_2 p$	Security	t/Arity	Rounds	Time	Data	Attack type
GMiMC$_{\mathsf{erf}}$	BC	128	128	4	60	2^{92}	2^{54}	KR
GMiMC$_{\mathsf{erf}}$	Permutation	61	128	12	102	2^{48}	–	ZS [8]
GMiMC$_{\mathsf{erf}}$	Permutation	125	256	14	206	2^{125}	–	ZS [8]
GMiMC$_{\mathsf{crf}}$		256	256	4	100	2^{159}	2^{96}	KR
GMiMC$_{\mathsf{erf}}$Hash	Hash	254	127	1 : 4	60	2^{119}	–	Coll
GMiMC$_{\mathsf{erf}}$Hash	Hash	61	128	4 : 12	52	2^{83}	–	Coll [8]

Table 4: Number of attacked rounds for specific GMiMC instances of GMiMC block ciphers and hash functions. For hash function the arity denotes the ratio of no. of branches used for hash output and total number of branches(t) in UFN.

comparison of our collision attack with the collision attack in [8] (in terms of complexity or number of rounds attacked) is not meaningful. For the attacks on GMiMC block ciphers, we provide key recovery attack on (reduced) 60-round GMiMC$_{\mathsf{erf}}$ over a prime field of size 128 bits. On the other hand a zero-sum(ZS) distinguisher on the full GMiMC(ERF) permutation (with fixed key) over $\mathbb{F}_q$ with $\log q = 125$ is proposed in [8].

References

1. Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 191–219. Springer, 2016.
2. Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for mpc, and more. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II*, volume 11736 of *Lecture Notes in Computer Science*, pages 151–171. Springer, 2019.
3. Martin R. Albrecht, Lorenzo Grassi, Leo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for mpc, and more. Cryptology ePrint Archive, Report 2019/397, 2019. `https://eprint.iacr.org/2019/397`.
4. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.
5. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques,*

Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

6. Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426, 2019. `https://eprint.iacr.org/2019/426`.
7. Tomer Ashur and Siemen Dhooghe. Marvellous: a stark-friendly family of cryptographic primitives. Cryptology ePrint Archive, Report 2018/1098, 2018. `https://eprint.iacr.org/2018/1098`.
8. Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. Out of oddity – new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. Cryptology ePrint Archive, Report 2020/188, 2020. `https://eprint.iacr.org/2020/188`.
9. Xavier Bonnetain. Collisions on Feistel-MiMC and univariate GMiMC. Cryptology ePrint Archive, Report 2019/951, 2019. `https://eprint.iacr.org/2019/951`.
10. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *Fast Software Encryption*, pages 313–333, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
11. Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low anddepth and few ands per bit. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 662–692, 2018.
12. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. `https://eprint.iacr.org/2019/458`.
13. Thomas Jakobsen and Lars R. Knudsen. The Interpolation Attack on Block Ciphers. In *In Fast Software Encryption*, pages 28–40. Springer-Verlag, 1997.
14. V. K. Leont'ev. Roots of random polynomials over a finite field. *Mathematical Notes*, 80(1):300–304, Jul 2006.
15. Chaoyun Li and Bart Preneel. Improved Interpolation Attacks on Cryptographic Primitives of Low Algebraic Degree. Cryptology ePrint Archive, Report 2019/812, 2019. `https://eprint.iacr.org/2019/812`.
16. Arnab Roy Lorenzo Grassi, Christian Rechberger. Gmimcs new key schedule. personal communication, 8 2019.
17. Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient fhe with low-noise ciphertexts. Cryptology ePrint Archive, Report 2016/254, 2016. `https://eprint.iacr.org/2016/254`.
18. Starkware. STARK-Friendly Hash Challenge. Website, 8 2019. `https://starkware.co/hash-challenge/`.
19. H.-J. Stoss. The complexity of evaluating interpolation polynomials. *Theoretical Computer Science*, 41:319–323, 1985.
20. Bing Sun, Longjiang Qu, and Chao Li. New cryptanalysis of block ciphers with low algebraic degree. In Orr Dunkelman, editor, *Fast Software Encryption*, pages 180–192, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
21. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*, YYYY. `https://www.sagemath.org`.

22. Joachim Von Zur Gathen and Jürgen Gerhard. *Modern Computer Algebra.* Cambridge university press, 2013.

A Proofs

A.1 Proof of Proposition 1

Proof. Suppose that Proposition 1 holds for fix r. Applying one more round $r+1$ yields the following, according to eq. (1):

$$\begin{aligned} \sigma_r &= f(P_0^{(r)} + k_r) \\ (P_0^{(r+1)}, P_1^{(r+1)}, P_2^{(r+1)}) &= (P_1^{(r)} + \sigma_r, P_2^{(r)} + \sigma_r, P_0^{(r)}) \end{aligned}$$

Developing σ_r by the binomial theorem we get the following:

$$\begin{aligned} \sigma_r &= (P_0^{(r)} + k_r)^d + \sum_{i=0}^{d-1} a_i (P_0^{(r)} + k_r)^i \\ &= \big(x^{d^{r-2}} + d^{r-3}(a_{d-1} + d\beta)x^{d^{r-2}-1} + \cdots + a_0 + k_r)^d + \sum_{i=0}^{d-1} a_i (P_0^{(r)} + k_r)^i \\ &= (x^{d^{r-2}})^d + d \cdot d^{r-3}(a_{d-1} + d\beta)x^{d^{r-2}-1}(x^{d^{r-2}})^{d-1} + \cdots + (d^{r-3}(a_{d-1} + d\beta)x^{d^{r-2}-1})^d \\ &\quad + \cdots + a_0^d + \cdots + k_r^d + \sum_{i=0}^{d-1} a_i (P_0^{(r)} + k_r)^i \\ &= x^{d^{r-1}} + d^{r-2}(a_{d-1} + d\beta)x^{d^{r-1}-1} + \cdots + a_0 \end{aligned}$$

By the assumption of the induction, $\mathsf{deg}(P_2^{(r)}) \leqslant \mathsf{deg}(P_1^{(r)}) \leqslant d^{r-2}$. Thus the degree of σ_r dominates, which leads us to the proofs first conclusions.

$$\begin{aligned} \mathsf{deg}(P_0^{(r+1)}) = \mathsf{deg}(P_1^{(r+1)}) &= \mathsf{deg}(\sigma_r) = x^{d^{r-1}} \\ \mathsf{coeff}(P_0^{(r+1)}, x^{d^{r-1}}) = \mathsf{coeff}(P_1^{(r+1)}, x^{d^{r-1}}) &= \mathsf{coeff}(\sigma_r, x^{d^{r-1}}) = 1 \end{aligned}$$

Since, by assumption, $d \geqslant 3$ and $r \geqslant 4$, it holds that $d^{r-1} - 1 > d^{r-2}$. The coefficients of the second highest term in $P_0^{(r+1)}$ and $P_1^{(r+1)}$ are thus solely contributed by σ_r. This leads us to the proofs last conclusion.

$$\mathsf{coeff}(P_0^{(r+1)}, x^{d^{r-1}-1}) = \mathsf{coeff}(P_1^{(r+1)}, x^{d^{r-1}-1}) = \mathsf{coeff}(\sigma_r, x^{d^{r-1}-1}) = d^{r-2}(a_{d-1} + d\beta)$$

A.2 Proof of Proposition 2

Proof. Because of the position of the variable x, $(t-1)$ many "swappings" of branches need to be performed before x becomes part of the input to a round function. Each round of the UFN performs exactly one such swap. Thus, we observe that x does not contribute to σ_i for the first $(t-2)$ rounds, i. e. $\mathsf{deg}(\sigma_i) = 0$

for $i < t-1$. In the $(t-1)$-st round, we have the following:

$$\begin{aligned}
\sigma_{t-1} &= f(x + \sigma_0 + \cdots + \sigma_{t-2} + k_{t-1}) \\
&= f(x+\beta) \\
&= (x+\beta)^d + a_{d-1}(x+\beta)^{d-1} + \sum_{i=0}^{d-2} a_i (x+\beta)^i \\
&= x^d + d\beta x^{d-1} + \cdots + d\beta^{d-1}x + \beta^d \\
&\quad + a_{d-1}x^{d-1} + a_{d-1}(d-1)\beta x^{d-2} + \cdots + a_{d-1}(d-1)\beta^{d-2}x + a_{d-1}\beta^{d-1} \\
&\quad + \sum_{i=0}^{d-2} a_i (x+\beta)^i \\
&= x^d + (a_{d-1} + d\beta)x^{d-1} + \cdots + a_0
\end{aligned}$$

Now, we can use induction over r much in the same way as in Proposition 1.

A.3 Proof of Proposition 4

Proof. After 1 round, we have

$$\sigma_0 = f(\sum_{j=1}^{t-1} b + k_0) = f((t-1)b + k_0)$$

$$(P_0^{(1)}, \ldots, P_{t-2}^{(1)}, P_{t-1}^{(1)}) = (b, \ldots, b, x + \sigma_0)$$

After 2 rounds, we have

$$\sigma_1 = f(\sum_{j=1}^{t-2} b + x + \sigma_0 + k_1) = f(x+\beta)$$

$$(P_0^{(2)}, \ldots, P_{t-2}^{(2)}, P_{t-1}^{(2)}) = (b, \ldots, x + \sigma_0, b + \sigma_1)$$

Expanding σ_1 by the binomial theorem yields the following:

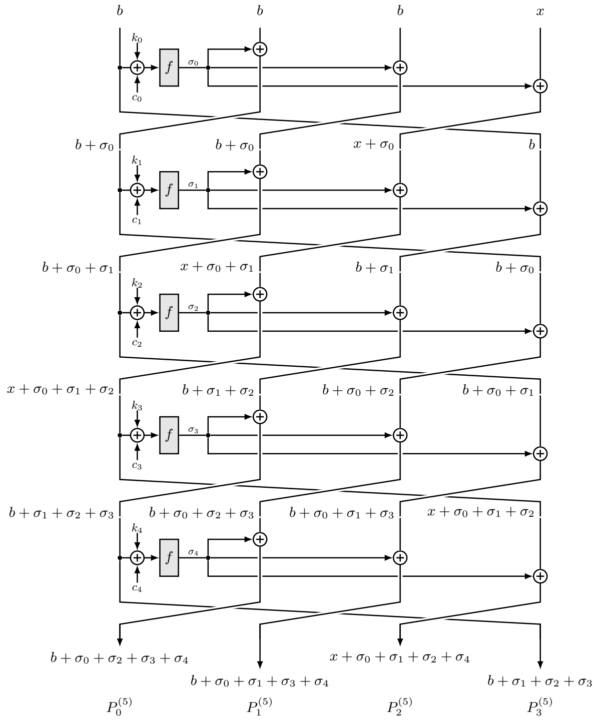
$$\begin{aligned}
\sigma_1 &= f(x+\beta) \\
&= (x+\beta)^d + a_{d-1}(x+\beta)^{d-1} + \sum_{i=0}^{d-2} a_i (x+\beta)^i \\
&= x^d + d\beta x^{d-1} + \cdots + \beta^d \\
&\quad + a_{d-1}x^{d-1} + a_{d-1}(d-1)\beta x^{d-2} + \cdots + a_{d-1}\beta^{d-1} + \sum_{i=0}^{d-2} a_i (x+\beta)^i \\
&= x^d + (a_{d-1} + d\beta)x^{d-1} + \cdots + a_0
\end{aligned}$$

After expanding σ_1, an induction over r like in Proposition 1 finishes the proof.

B Reduced Round Instances of $\text{UFN}_{\mathsf{erf}}$and $\text{UFN}_{\mathsf{crf}}$

We depict the progression of an input (b, b, b, x) through 5 rounds of $\text{UFN}_{\mathsf{erf}}$ with 4 branches in fig. 2. In fig. 3, we show the progression of an input variable through 4 rounds of $\text{UFN}_{\mathsf{erf}}$ with 3 branches and variable input in the rightmost branch as well as $\text{UFN}_{\mathsf{crf}}$ with 3 branches and variable input in the leftmost branch.

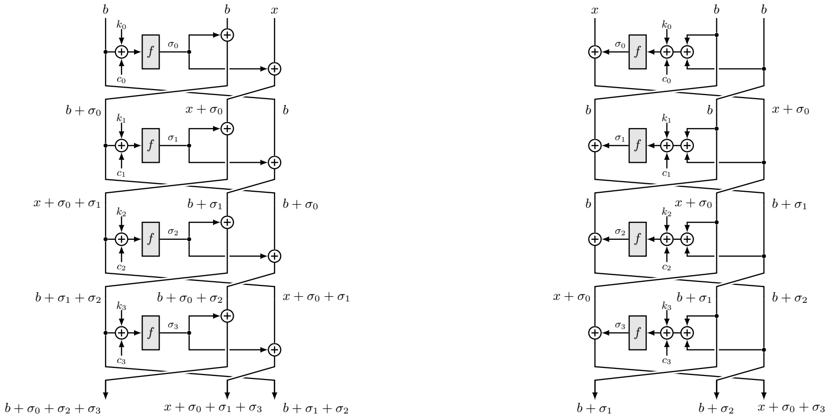
Fig. 2: Example of summands being added in a $\text{UFN}_{\mathsf{erf}}$



C Plots and runtime summaries for GMiMCHash experiments

In tables 5 and 6 we summarize the running times for the experiments from section 7.1 for $5 \leqslant r \leqslant 7$ and $3 \leqslant t \leqslant 6$. The running times for reconstruction of the polynomial and root finding are reported alongside the total running times.

In fig. 4, the number of second preimages found in our experiments is plotted. Similarly, fig. 5 visualizes the number of collisions found. In the subfigures,



(a) Branch development in a $\text{UFN}_{\mathsf{erf}}[p, 4, 3]$.

(b) Branch development in a $\text{UFN}_{\mathsf{crf}}[p, 4, 3]$.

Fig. 3: $\text{UFN}_{\mathsf{erf}}[4,\ 3]$ and $\text{UFN}_{\mathsf{crf}}[4,\ 3]$ with rightmost and leftmost branch input as variable respectively.

different numbers of branches t are depicted. Each subfigure shows, for different numbers of rounds r on the x-axis, the number of additional preimages or collisions on the y-axis found over the 100 randomized experiments. For example, when regarding $t = 3$ branches in fig. 4a, for the GMiMCHash instance instantiated with $\text{GMiMC}_{\mathsf{erf}}$ with $r = 6$ rounds, there were 14 of our 100 experiments in which we found 2 preimages, and 9 in which we found 3 preimages. A red bar signifies that no preimage (or no collision for fig. 5) was found, while the green bars indicate at least one second preimage (or collision, respectively) found, i. e. a successful attack.

D Roots of Random Polynomials over a Finite Field

In order to validate our failure rate of section 7.1, we calculate the probability that a random polynomial of degree d has no roots in a specific finite field. A formula t_d for the number of polynomials of degree d over finite field $\mathbb{F}_q$ that have no root in $\mathbb{F}_q$ is given in lemma 1 in [14] and reproduced in eq. (13).

$$t_d = \sum_{i=0}^{d} (-1)^i \binom{q}{i} q^{d-i} \tag{13}$$

The total number of polynomials of degree d over $\mathbb{F}_q$ is q^d. Our parameters are $q = 99999989$ and $d = 3^i$ with $3 \leqslant i \leqslant 7$. This results in a probability of a random polynomial not having any root in $\mathbb{F}_q$ of ${}^{t_d}/_{q^d} \approx 36.8\%$ for any i in the given interval.

	$r = 5$	$r = 6$	$r = 7$
construct poly	0.0003	0.0003	0.0003
root finding	0.0005	0.0005	0.0004
total	0.0008	0.0009	0.0008

(a) $t = 3$

	$r = 5$	$r = 6$	$r = 7$
construct poly	0.0019	0.0019	0.0019
root finding	0.0026	0.0029	0.0032
total	0.0046	0.0049	0.0052

(b) $t = 4$

	$r = 5$	$r = 6$	$r = 7$
construct poly	0.0233	0.0235	0.0229
root finding	0.0263	0.0267	0.0268
total	0.0498	0.0503	0.0498

(c) $t = 5$

	$r = 5$	$r = 6$	$r = 7$
construct poly	0.2966	0.2940	0.2954
root finding	0.3018	0.3016	0.3025
total	0.5999	0.5971	0.5996

(d) $t = 6$

Table 5: Observed average running times in milliseconds for finding second preimages of GMiMCHash using GMiMC$_{\mathsf{erf}}[p, r, t]$. ($n = 100$ per column)

	$r = 5$	$r = 6$	$r = 7$
construct poly	0.0006	0.0006	0.0006
root finding	0.0004	0.0004	0.0004
total	0.0010	0.0010	0.0010

(a) $t = 3$

	$r = 5$	$r = 6$	$r = 7$
construct poly	0.0037	0.0037	0.0042
root finding	0.0028	0.0030	0.0038
total	0.0065	0.0067	0.0080

(b) $t = 4$

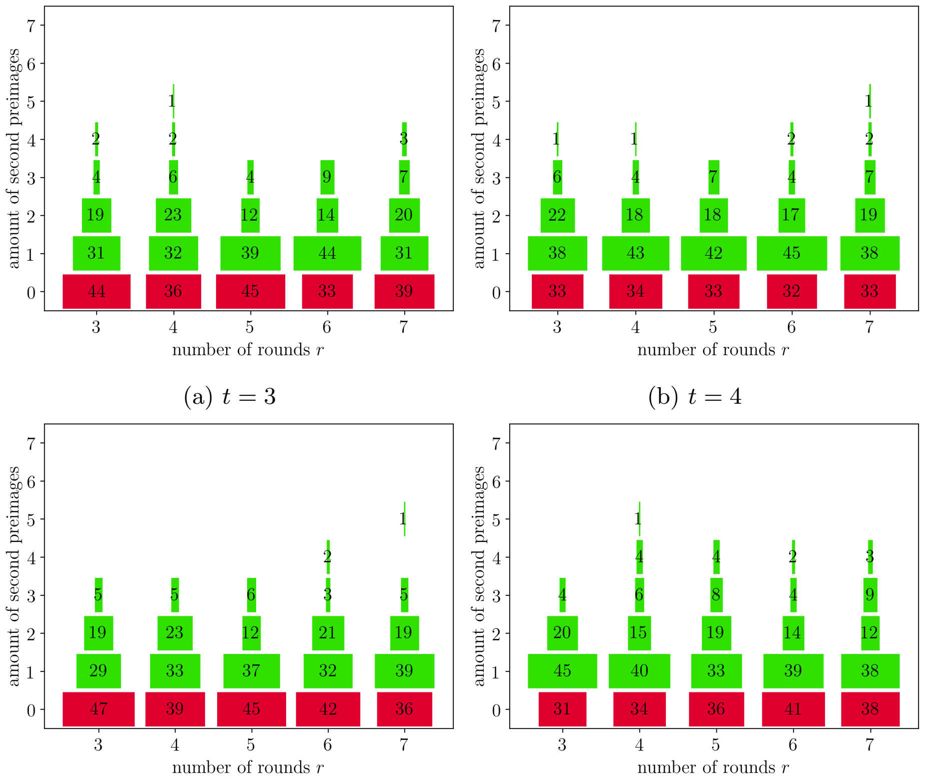
	$r = 5$	$r = 6$	$r = 7$
construct poly	0.0458	0.0457	0.0458
root finding	0.0275	0.0280	0.0269
total	0.0733	0.0738	0.0728

(c) $t = 5$

	$r = 5$	$r = 6$	$r = 7$
construct poly	0.6015	0.6012	0.6801
root finding	0.3116	0.3063	0.3536
total	0.9131	0.9075	1.0338

(d) $t = 6$

Table 6: Observed average running times in milliseconds for collision finding of GMiMCHash using GMiMC$_{\mathsf{erf}}[p, r, t]$. ($n = 100$ per column)



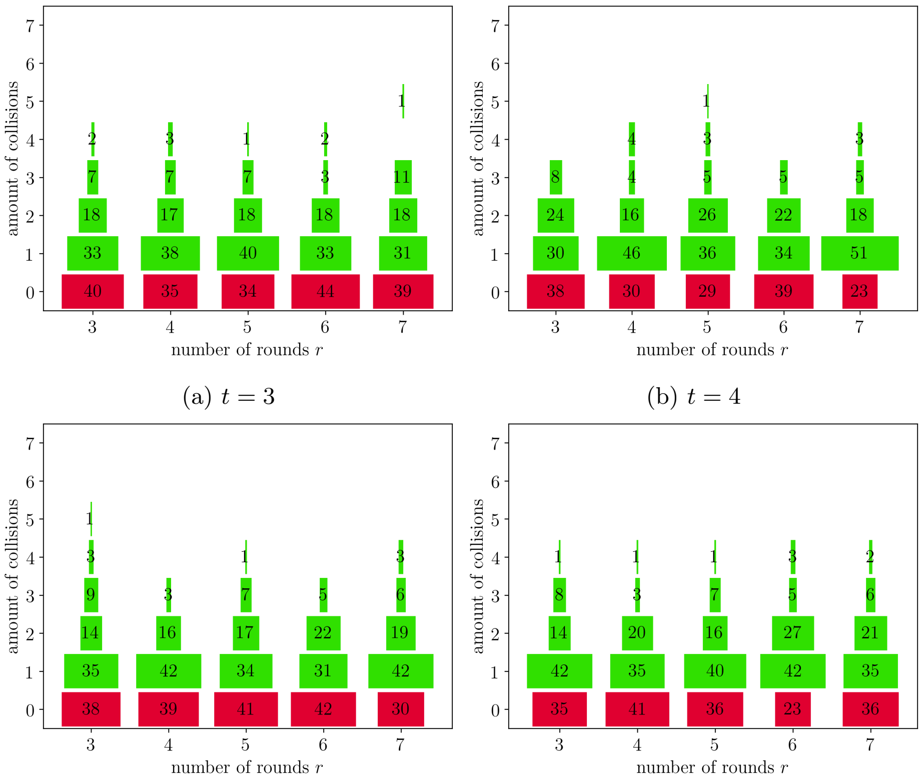
(a) $t = 3$

(b) $t = 4$

(c) $t = 5$

(d) $t = 6$

Fig. 4: Number of second preimages found for various numbers of rounds with GMiMCHash using $\text{GMiMC}_{\mathsf{erf}}[p, r, t]$. ($n = 100$ per given (r, t))



(a) $t = 3$

(b) $t = 4$

(c) $t = 5$

(d) $t = 6$

Fig. 5: Number of collisions found for various numbers of rounds with GMiMCHash using GMiMC$_{\mathsf{erf}}[p, r, t]$. ($n = 100$ per given (r, t))