Post Quantum Lightweight OWF Candidates
Based on Theoretically Secure Primitives: Xors, Error Detection Codes, Permutations, Polynomials, Interaction, and Nesting

Paweł Cyprys, Shlomi Dolev, and Oded Margalit

Ben-Gurion University of the Negev, Israel, Beer-Sheva

Abstract. Our research focuses on designing efficient commitment schemes by drawing inspiration from (perfect) information-theoretical secure primitives, e.g., the one-time pad and secret sharing. We use a random input as a mask for the committed value, outputting a function on the random input. Then, couple the output with the committed value xored with the random input folded (half of the input xored with the other half of the) random input.

First, we explore the potential of leveraging the unique properties of the one-time pad to design effective one-way functions. Our methodology applies the exclusive-or (xor) operation to two randomly chosen strings. To address concerns related to preimage mappings, we incorporate error detection codes. Additionally, we utilize permutations to overcome linearity issues in the computation process. Feistel networks are employed to ensure super pseudo-random permutation using the (random string) input (that serves as the commitment mask) and also as the encryption key.

We propose integrating a secret-sharing scheme based on a linear polynomial to mitigate possible collisions. Lastly, we explore the possibility of nesting one-way functions as a countermeasure against potential backdoors.

The resulting commitment schemes are efficient, in particular, have fewer layers than the standard cryptographic hash functions, such as SHA, and may fit the NIST effort for lightweight IoT cryptography (e.g., ASCON [DEMS21]).

1 Introduction and Related Work

We propose exploring computationally efficient one-way functions on randomly chosen inputs that can be an alternative to Secure Hash Algorithms (SHA) [Han11]. These functions should resist preimage and collision attacks, providing improved security for commitments and signatures, such as Lamport's signature [Lam79]. Relying solely on block-cipher-based functions like SHA may lead cryptanalysts to focus their efforts on breaking these functions, as demonstrated by the vulnerabilities found in MD4 [Leu08] and MD5 [SA09], not to mention the potential existence of backdoors [FJM18]. To enhance security and mitigate risks, we propose a new approach where information theoretically secure primitives such as one-time pads, secret sharing, and permutations are composed in different ways, including nesting, as heuristics for hardening inversions.

Our aim is to explore a range of computationally efficient one-way functions that (may utilize the multi-core architecture) can expand the choices available to implementers. Our schemes do not rely on number theory, which is vulnerable to quantum algorithms like Shor's algorithm [Sho97] that can compromise the security of DH and RSA [Sho94], surpassing the quadratic speedup provided by Grover's algorithm [Gro96]. Commitments based on one-way functions[1] find applications in various scenarios, including Zero Knowledge Proofs (ZKP) [BM89, GMR89], where one-way functions are used as commitment primitives. While cryptographic hash functions like the SHA family are designed to handle long inputs (e.g., files) and are expected to have collisions due to the pigeonhole principle, our main focus in this paper is on inputs of the same length or smaller than the output. We suggest using Merkle trees to handle longer inputs, handling the left and right child of a tree differently prior to applying our OWF candidate on their xor, namely, reversing the bits of the left child output before xoring the outputs of the left and right children. Where reversing a binary string of length l involves permuting the string such that the i-th bit becomes the $(l-i+1)$-th bit. [GLN11].

We explore techniques to improve the one-way properties of existing one-way function candidates by utilizing xor operations in the style of a one-time pad. Recently, xor operations among essential portions of instances of the original one-way functions were proposed [DD20, CDM22, CCD23]. The objective is to mimic xor with a one-time pad to ensure the masking of the success criteria of an instance (e.g., the sum in a subset-sum instance) while limiting the possible number of preimages. In commitment schemes, it is undesirable to have multiple fitting preimages or collisions, as the committer could select a preimage from the colliding set when revealing the commitment. Additionally, if there are numerous colliding preimages, the task of reversing the output of the one-way function candidate may become relatively easy. Univalent of the one-way function is a desired property. In contrast to [DD20, CDM22, CCD23], the focus is on the perfect information-theoretically secure primitives rather than enhancing already existing one-way functions (e.g., subset-sum) with such primitives.

2 Overview

To simplify the structure and arguments related to the one-way functions, we suggest starting with a totally random and uniformly chosen input string, *input*, to the function to obtain the *output*. Thus, avoiding the possible guess of the commitment value and checking the guess by applying the one-way function, as the structured functions are publicly known.

The actual commitment value, *commit-value*, is XORed with a folded version of the random input *input*. Namely, outputting both *output* and the first half of

[1] Note that provable one-way function implies that $P \neq NP$.

the *input*, *fh-input*, XORed with the second half of the *input*, *sh-input*, and the *commit-value* (assuming the *input* random string is assumed to be double the length of the *commit-value*).

In summary, the commitment is a pair *output* and *fh-input* $\oplus$ *sh-input* $\oplus$ *commit-value*, where $\oplus$ denotes bitwise xor.

Obviously, folding more than twice enlarges the possible number of inputs to be checked, even when the commit value is guessed. On the other hand, more foldings may yield more possible collisions.

• **Xors.** We aim to utilize the one-time pad's provable information-theoretically secure principle. In this study, we explore the potential of leveraging the inherent properties of the one-time pad to design one-way functions. We randomly select two strings, s_1 and s_2, each consisting of n bits. It is important to note that the result of performing a bitwise xor operation on s_1 and s_2, denoted as r_{12}, encompasses all possible combinations of s_1 and compatible counterparts of s_2 that yield r_{12}. As a result, reversing the process and obtaining r_{12} is relatively easy and leads to a multitude of possible answers (collisions), which grows exponentially with the lengths of s_i ($n = |s_i|$).

• **Error Detection Codes.** To address the issue of excessive collisions and improve the difficulty of reversing the function, we propose using error detection codes such as Cyclic Redundancy Check (CRC), Hamming codes, Reed-Solomon codes, and binary Goppa codes. For each s_i, we introduce an error detection code edc_i. The computation of r_{12} is then performed as follows: $(s_1 \circ edc_1) \oplus (edc_2 \circ s_2)$, where $\circ$ denotes concatenation. To facilitate discussion, we set the length of the error detection codes equal to that of the original strings they represent, i.e., $|edc_i| = |s_i|$.

• **Permutations.** The design naively attempts to utilize s_1 (and s_2) as a one-time pad for edc_2 (and edc_1, respectively). However, we demonstrate that in cases where the error detection code is linear, there exist polynomial time algorithms that can invert r_{12} and recover s_1 and s_2 with relative ease. To cope with the (reversible) linearity of error detection codes, we suggest using permutations, permuting edc_i by the values of s_j. For ease of discussion, we suggest using $2\lg(n)+1$ pairs s_1^i, s_2^i, namely, s_1^1, s_2^1, s_1^2, s_2^2, s_1^3, s_2^3, $\cdots$, $s_1^{2\lg(n)+1}$, $s_2^{2\lg(n)+1}$ compute for each of s_1^i, s_2^i the value $r_i = (s_1^i \circ s_2^i) \oplus \pi_{s_1^1, s_2^1, \cdots s_1^{i-1}, s_2^{i-1}, s_1^{i+1}, s_2^{i+1}, \cdots s_1^{2\lg(n)+1}, s_2^{2\lg(n)+1}}(edc_1^i \circ edc_2^i)$. Where $\pi_{s_1^1, s_2^1, \cdots s_1^{i-1}, s_2^{i-1}, s_1^{i+1}, s_2^{i+1}, \cdots s_1^{2\lg(n)+1}, s_2^{2\lg(n)+1}}$ is a permutation chosen in lexicographic order among all the $(2n)!$ possible permutations (note that the number of bits used should be $\lg((2n)!) < \lg((2n)^{2n}) < 2n\lg(2n)$ which is smaller than the suggested $2n(2\lg(n)+1)$ bits). The mapping of a binary number to a permutation can be based on, e.g., [DLH13], using mod $(2n)!$ as fits.

• **Permutation-Holographic[2] via Feistel Network.** We choose four or (even seven) rounds of Feistel networks as [LR88] ([Knu02a]) shows that they imply

[2] A function is considered holographic if every bit in the output is a (different, and even orthogonal) function of all the input bits. Moreover, the output bits encode all the input bits. The term is inspired by holography, where each part of a hologram contains information about the whole image.

(super) pseudorandom permutation. Using the input to the hash also to serve as the key allows a better probability of collision-free results, as encryption must support decryption and cannot allow collisions. Thus, Feistel (x, x), just like AES(x, x), has the potential to yield pairwise independence, as our experiments testify. Still, computing Feistel (x, x) can be more efficient in relation to AES (or SHA), which uses more than four/six rounds.

Apparently, we cannot use the left portion as the key for the next round in all rounds of the Feisel network, as if we do so, one can decrypt the preimage (using the decryption procedure of the Feistel network) as the key used in each round appears on the left side. Thus, we suggest using the right portion in the last round, avoiding collisions until then and, at last, avoiding the exposure of the decryption key. We also suggest to use k independent hash function $H(x) = a_0 + a_1 x + \ldots a_{k-1} x^{k-1}$, such that $a_0, a_1, \ldots, a_{k-1}$ are k portions of the input x. $H(x)$ is used as a part of the definition of the F function used in the rounds of the Feistel network circuit. In particular, $H(x)$ can be applied to the left portions of all but the last round and applied to the right portion of the last round to define the permutation on the other portion. If the number of bits of $H(x)$ does not suffice, the result of $H(x)$ can be concatenated to form the required number of bits.

• **Polynomials.** We recommend implementing secret-sharing schemes to avoid the scenario where multiple preimages, known as collisions, lead to the same function output. Secret sharing is another (beyond one-time-pad) very useful, proven perfect information theoretical secure primitive. A secret is represented through distributed secret shares among participants, wherein reconstructing the original secret requires possessing more than a specified threshold number of shares. In this scheme, the value r_i may have multiple pre-images, but we still manage to restrict collisions. The commitment value is determined by the intersection of a line (or polynomial) with the y-axis, denoted as x. An interactive commitment approach is proposed to improve security, where the committing party receives random x-values. The Fiat Shamir technique is suggested to replace interaction. This process ensures that even if only two values on the line do not collide, the commitment is still unique and can be attributed to a single possible value.

• **Interaction and Nesting.** We further suggest a nesting of one-way functions in which the committing party is instructed first to use a certain one-way function to eliminate planned collisions; then, the committing party uses the output of the given one-way function as an input for her choice of a one-way function to eliminate possible backdoors in the first determined one-way function. Both parties should agree (and possibly verify by probabilistic sampling) that the suggested functions imply a small number of collisions. For a more comprehensive background on one-way functions and related applications, see, e.g., [IN96, HILL99, DD20, CDM22, CCD23].

Paper roadmap. Section 3 develops the reasoning for the need for permutation beyond xors (in the style of mutual one-time pads). The extension of that section also describes a holographic approach to bit permutation within commitment.

In Section 4, we investigate the application of Feistel networks combined with our idea. Section 5 introduces using (linear) polynomials to cope with possible collisions. Section 5 develops a formal analysis of the described idea. Section 6 uses the interaction between the committer and the verifier to cope with potential planned collision (by the committer) and planned backdoors (by the verifier). Finally, concluding remarks appear in Section 7.

Throughout the paper, we illustrate the proposed concepts using toy examples. For the convenience of readers, the implementations required to replicate these examples can be found in [CDM23]. The software was implemented using the SageMath computational software environment [The23].

3 XORS, Error Detection Codes, and Permutations

We illustrate that without the use of permutation, the application of xor, which aims to emulate Shannon's "one-time pad" concept (as described in [Sha49]), by mutually masking the error detections of s_1 and s_2, can be easily inverted in polynomial time.

To demonstrate the potential ease of inversion, we consider the following specific example: s_1 and s_2 are randomly chosen four-bit strings. We utilize a standard CRC-4-ITU algorithm (with the polynomial representation $x^4 + x + 1$ as defined in [RS91]) to compute crc_1 for s_1 and crc_2 for s_2, where each crc value consists of four bits. Subsequently, we compute $f(s_1, s_2) = (s_1 \circ crc_1) \oplus (crc_2 \circ s_2)$.

Due to the linearity of the process, we observe that $f(s_1, s_2) = (s_1 \circ s_2) \cdot A$, where A is a square matrix that implements the function f.

Note that there exists a one-to-one linear mapping between the values of s_1 and s_2 and $f(s_1 \circ s_2)$, and it is feasible to construct a matrix that computes f.

Furthermore, f may be invertible, as f^{-1} can retrieve $s_1 \circ s_2$ from the output of $f(s_1 \circ s_2)$. The inverse function f^{-1} can be computed in polynomial time by utilizing the inverse matrix A^{-1}, such that $s_1 \circ s_2 = f(s_1, s_2) \cdot A^{-1}$.

Appendix A presents a specific numerical example illustrating the straightforward nature of such inversion.

Incorporating permutations.

To improve the reconstruction of critical parts, such as s_1 and s_2, we can extend the self-masking technique, presented in [CDM22] with CRC codes by incorporating permutation using permutation indices. This approach allows us to define an actual permutation.

In our example, we have presented a construction where a binary array is generated based on a string $s = s_1 \circ s_2$, with a binary length of $|s| = 8$ (suitable for representing a single character in ASCII encoding). The first three bits of the binary array are utilized to determine the parameter p, while the next three bits determine the parameter q. These parameters, p and q, are then used to define a permutation denoted as π. The permutation function π is employed to map the elements of the binary string to a new instance of the binary string with permuted elements. Specifically, the elements (bit values) of the computed

$crc_1 \circ crc_2$ are swapped based on the indexes defined by p and q using this permutation mapping. Namely, given two strings $s_1 = a_1, a_2, a_3, a_4$, $s_2 = b_1, b_2, b_3, b_4$, compute $crc_1 = c_1, c_2, c_3, c_4$ for s_1, and compute $crc_2 = d_1, d_2, d_3, d_4$ for s_2. Then, consider the sequence $crc = crc_2 \circ crc_1 = d_1, d_2, d_3, d_4, c_1, c_2, c_3, c_4 = e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8$, permute elements in a way that swapes e_{p+1} with e_{q+1}. The Blackbox then xors $s = a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4$ with the permuted crc.

Permutations example. Let us consider the following (toy) example that illustrates the ineffectiveness of polynomial-time inversion when permutation is applied in the BlackBox.

For instance, let us define $s_1 = 1001$ and $s_2 = 1010$. The resulting BlackBox output for s_1 and s_2 is denoted as r:

$$s_1 \circ s_2 = [1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0], \quad r = [0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0]$$

$$B = \begin{bmatrix} (s_1^1 \circ s_2^1) \oplus \pi_{s^1}[crc_2^1 \circ crc_1^1] \\ (s_1^2 \circ s_2^2) \oplus \pi_{s^2}[crc_2^2 \circ crc_1^2] \\ \vdots \\ (s_1^8 \circ s_2^8) \oplus \pi_{s^8}[crc_2^8 \circ crc_1^8] \end{bmatrix} \qquad B = \begin{bmatrix} 0\,0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0\,0 \\ 1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1\,1\,0\,1 \\ 0\,0\,0\,0\,0\,1\,0\,1 \\ 0\,0\,0\,0\,1\,0\,1\,0 \end{bmatrix}$$

We construct the matrix B to align with the new Blackbox, which incorporates a limited permutation and enables the computation of any row of the identity matrix I. This capability directly stems from the construction of matrices B (and A).

However, attempting to employ the same linear technique used in Appendix A by substituting matrix A with B proves unsuccessful due to the nonlinearity of the permutation.

In particular, the result of the Blackbox for the input $s_1 \circ s_2 = [1\ 0\ 0\ 1\ 1\ 0\ 1\ 0]$ is $r = [0\ 0\ 1\ 0\ 1\ 1\ 1\ 0]$ while the multiplication of $s_1 \circ s_2 = [1\ 0\ 0\ 1\ 1\ 0\ 1\ 0]$ by B yields a different vector of bits. When we perform xor operation over the rows $0, 3, 4$ and 6 of B we get the vector: $[0\ 1\ 0\ 0\ 0\ 1\ 1\ 1] \neq r = [0\ 0\ 1\ 0\ 1\ 1\ 1\ 0]$.

Based on the example above, it becomes evident that constructing matrix B according to the permuted Blackbox does not enable the utilization of B (as with A) to compute all the results of the new permuting Blackbox.

Although the limited permutation may reduce the number of combinations to be examined when attempting to reverse the function, we now aim to improve the permutation operation to achieve a complete random permutation based on bits from other instances.

Extension to Merkle Tree. Combining holographic permutations with a Merkle tree, we efficiently handle large commitments. Each leaf processes a random sequence segment, using xoring and bit reversals (of the left child as described earlier) to create a holographic Merkle root, offering a novel hash function alternative. The commitment combines the value with a packed sequence version, linked to the Merkle root, and publishes both. We refine permutation efficiency, reducing lookup table overhead through bit manipulation and possible precomputed tables.

4 Using Feistel Networks for XORs and Permutations

Feistel networks are fundamental in cryptography due to their invertible structure, ensuring collision-free outputs when using a fixed symmetric key. In the context of one-way functions (OWFs), no secret key exists. Therefore, we propose using portions of the randomly chosen input string as both the input and the key.

Note that the authors of [Knu02b] showed that the probability a randomly chosen function from F_n (the set of all functions mapping $\{0,1\}^n$ to itself) is a permutation is $\frac{2^n!}{2^{n2^n}}$, which, using Stirling's approximation, is roughly $\sqrt{2\pi 2^n}/e^{2^n}$ for large n. This probability decreases rapidly with n. For example, for $n = 6$, the probability is about 2^{-88}. If a randomly chosen function from F_n is not a permutation, it is not bijective and may have collisions. Using permutations is crucial as they ensure the encryption process is reversible.

We investigate the behaviour of collision percentages in Feistel networks using part of the input as a key. Experiments under different bit lengths provide theoretical insights and empirical results, aiming to understand collision conditions and frequencies, relating these findings to the Coupon Collector Problem [FGT92].

A Feistel network can be described in the following way: Let P be the plaintext block divided into two halves: L_0 (left half) and R_0 (right half). For $i = 1$ to n (where n is the number of rounds), the round transformations are:

$$L_i = R_{i-1}, \quad R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

where:F is the round function and K_i is the subkey for the i-th round. The ciphertext C after n rounds is given by (L_n, R_n).
Decryption Process The decryption process uses the same structure but applies the subkeys in reverse order.

Our implementation uses a specific strategy for the round function and key selection.

- **First Three (or Six) Rounds** We alternate using the left and right halves of the input as the key for generating permutations in the round function F. The half used as the key is repeated if necessary to match the required length.
- **Last Round** We use the right half of the input as the key for the round function F. Avoiding exposure of the left half, which we use as a key (for encryption and decryption).

Using only the left half initially does not introduce collisions as it acts as a decryption key, leaving collision introduction to the last layer.

Experiments processed data through Feistel networks with varying bit lengths (8 to 26 bits), using different strategies for F and key selection. Initially, no collisions occurred when F used a key from the left half (L), confirming invertibility. In our configuration, we derived a key from L for the first three rounds (or six) and from the right half (R) in the last round. F uses a "Unique Permutation", described in [DLH13], defining a permutation based on a binary string. If there are not enough bits, we copy and concatenated R to meet the bit requirement. This keeps the key unique and important for the first three rounds to avoid collisions. Collisions were observed, with the percentage increasing with bit length:

Bit Length (n)	**Collision Probability $C(n)$ (%)**
8	25.00
10	34.38
12	31.25
14	40.62
16	35.55
18	36.72
20	36.72
22	37.60
24	36.94
26	38.06

Table 1. Collision probabilities for different bit lengths

Unique Permutation and Extended Key. Further experiments extended the key with pseudo-random values and ensured unique permutations per round.
Collision Rates Stabilized around 36.79% (approximating $1 - \frac{1}{e}$), as predicted by probabilistic models.
Performance Comparison with AES We compared our Feistel network with AES in CBC mode using 128-bit inputs on a machine equipped with an Apple M1 Pro processor (ARM-based). The code was executed on a single thread using Python 3.12.4:

Algorithm	**Processing Time (seconds)**
Feistel Network	0.000268
AES-CBC	0.041304

Table 2. Processing Times for Different Algorithms

The Feistel network was significantly faster. However, this comparison is limited due to optimized AES libraries and hardware acceleration.

The observed collision percentages align with the Coupon Collector Problem (See, e.g., [MU05]). In large sets, the probability that a new trial does not yield a new item converges to $1 - \frac{1}{e} \approx 36.79\%$. Our empirical data supports this.

Our scheme offers IND-CCA (Indistinguishability under Chosen-Ciphertext Attack) security due to the random selection of x. The commitment for plaintext pt is generated by uniformly choosing x independently of pt, computing $F(x, x)$, where F represents our construction based on Feistel network, and then XOR-ing the folded x with pt, defined as $fold$, where the fold operation is defined as $fold(x) = \texttt{prefix}(\frac{|x|}{2}, x) \oplus \texttt{suffix}(\frac{|x|}{2}, x)$. This process ensures that the choice of plaintext pt does not reveal any information about the commitment. As long as $F(x,x)$ is IND-CCA secure, the final output $F(x,x)$, $fold(x) \bigoplus pt$ also maintains IND-CCA security. Note that our scheme benefits from using real random x in every invocation, basing the random property abstracted by a random oracle. However, different from the random oracle abstraction the outputs of our scheme can differ across invocations over the same input x. We believe these new characteristics maybe useful as a building block in future schemes constructions.

5 Polynomials for Collision Prevention

In this section, we propose a novel approach to address collisions that can be applied to other cryptographic hash functions, such as MD5. We utilize the concept of secret sharing, where the actual committed value is encoded using a polynomial, specifically a line.

Our new hash function employs a primitive cryptographic hash function to hash $f(0)$. The x values can range from 1 to m, where m is a chosen parameter. Alternatively, the verifying party can define the x values, creating a somewhat interactive commitment process. Knowing that the queries are planned to be in certain x's, the committer may choose polynomial parameters that yield meaningful collisions for the known set of x's.

Thus, we prefer to reveal the future x's beyond the first two that restrict the set of possible lines (assuming there are collisions for at least one of the first two x's) by choosing x's for which the committer cannot further choose upfront parameters for the commitment to yield colinearity of collisions. We hypothesize that the committing party will be unable to coordinate two lines from the collisions of these m hashed values in a way that encodes a different line and a distinct secret. In cases where the number of collisions for each value is limited, as in our suggested one-way function, we can demonstrate that the number of possible collisions diminishes towards zero. More details can be found in the Appendix C.

Our approach addresses MD5's vulnerabilities [SLDW07] by employing polynomials over finite fields, ensuring unique and secure commitment mapping and reducing collision risks.

To address this issue, we introduce a novel approach that leverages the mathematical properties of polynomials defined over finite fields to ensure the uniqueness of each commitment.

Commitment Representation. We represent the secret commitment as a point on the y-axis of a two-dimensional graph. This point lies on a straight line defined by the polynomial equation $y = ax + b$, where b is the value to be committed upon.

Hash Function Application. Rather than hashing the secret directly, we select multiple points along this line and hash their y-values. Note that this has an extra benefit, as at least two preimages should be revealed to reveal the actual committed value. The x-values for these points can be fixed in advance or chosen interactively during the verification process to improve security. One more option to determine the x values and eliminate the need for interaction is to

use a technique inspired by the Fiat-Shamir scheme [FS87a], where the previous results (related to previous points) on applying the one-way-function are xored (other functions can be applied instead) to yield the next x value(s).
Ensuring Uniqueness. The uniqueness of the commitment can be improved through strategic selection of x-values for hashing. After the initial points are disclosed, subsequent points are chosen such that any attempt to adjust the line to pass through these new points would fail unless the initial secret was accurate. This strategy leverages the geometric principle that two distinct lines cannot share more than one point unless they are identical. Moreover, by hashing the y-values of selected points on the line, we further utilize the polynomial's properties to guarantee that the hashed outputs are unique to the original secret value. This approach effectively minimizes the risk of collisions, offering a significant advantage over traditional hash functions like MD5.
Formal Analysis. Next, we elaborate on our cryptographic commitment scheme using a mathematical framework. We formally define essential constructs, including finite fields, polynomials, commitment schemes, and cryptographic hash functions, and analyze their interactions to establish the security properties of our protocol.
Finite Field ($\mathbb{F}_q$) A set of q elements with addition and multiplication operations defined, where every non-zero element has a multiplicative inverse.
Polynomial ($P(x)$): An expression of the form

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

where coefficients a_i are in $\mathbb{F}_q$. We consider only linear polynomials: $a_1 x + a_0$.
Commitment A hiding commitment scheme for a secret value $s \in \mathbb{F}_q$ is realized by computing a polynomial $P(x) = ax + s$, where a is a randomly chosen slope from $\mathbb{F}_q$. The commitment is the y-intercept $P(0) = s$.
Hash Function (H) A cryptographic hash function mapping arbitrary-length input data to a fixed-size output in $\mathbb{F}_q$. We assume that H is collision-resistant and its outputs are (close to) uniformly distributed in $\mathbb{F}_q$.
In the *Fiat-Shamir heuristic*, random oracles replace interaction in cryptographic protocols. For our polynomial commitments, we can use the hash function output (H) as a source of "randomness" to choose the (x)-coordinates of the points to be hashed. This eliminates the need for back-and-forth communication between the committer and the verifier.

Definition 1. *Let f_h be a hash function used in the commitment scheme. It takes an input value y_i and produces a hashed output. This output is used to determine the next input value in the sequence $x_{i+1} = f_h(y_i)$ and also serves as the commitment value $b = f_h(y_0)$.*

The next Theorem assumes a hash function f_h that has a maximum of d collisions (i.e., possible distinct preimages) for any given $f_h(y_i)$ output value, where d is significantly smaller than q, and the colliding value are assumed to be uniformly distributed in the field.

Theorem 1. *A committer in our scheme that generates a set of n points (for $n \geq 3$), where each point is of the form $(x_i, p(x_i))$, $p(x) = ax + b$ is a line with a random slope a and a y-intercept b representing the commitment value, exposing $(x_i, f_h(p(xi))$, has a negligible probability to output hashes that have preimages that lie on $c > 1$ lines in $\mathbb{F}_q$.*

Proof. Consider $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ to be a set of n distinct inputs. The input x_1 is a default value, and the subsequent inputs are determined (following the Fiat-Shamir technique) as follows:

$x_{i+1} = f_h(y_i)$ for $1 \leq i < n$, in case $x_{i+1} = x_j,\ j \leq i$, then x_{i+1} is the smallest value in the field that is different from every $x_j,\ j \leq i$.

Let $\mathcal{F}(\mathcal{X}) = \{(x_1, p(x_1)), (x_2, p(x_2)), \ldots, (x_n, p(x_n))\}$ be the corresponding set of points, where $p(x_i) = ax_i + b$.

For each input x_i, the committer reveals a value y_i that they claim to have hashed to obtain $f_h(y_i)$.

Our hash function f_h has a maximum of d collisions (i.e., possible distinct preimages) for any given $f_h(y_i)$ output value, where d is significantly smaller than q, and typically are assumed to be uniformly distributed in the field.

The first two points, $(x_1, p(x_1))$ and $(x_2, p(x_2))$, define at most d^2 lines.

For each subsequent point (x_i) (where $3 \leq i \leq n$), the probability that the hash value associated with x_i has a preimage that lies on more than one line of the at most d^2 lines that are defined by x_1 and the exposed value of hash related to x_1, and by x_2 and the exposed hash function related to x_2, is bounded by:

$$\left(\tfrac{d}{q}(d^2 - 1)\right)^n$$

This probability becomes negligible as q (the field size) is typically much larger than d in the original hash function. □

6 Nesting

In a scenario where a commitment scheme is used, allowing one side to choose the one-way function can lead to adversarial behaviour. Consider a situation where a gambler wants to commit to a specific colour (red or black) for a bet in a casino's roulette game before the ball stops rotating.

The gambler may have doubts about the casino potentially manipulating the outcome in favour of the unchosen colour, even if they commit to their chosen colour using a one-way function (where the colour is encoded using enough bits combined with a random nonce). The gambler can choose a one-way function that exhibits collisions, meaning both red and black can be preimages of the function's output. This poses a risk since the casino needs to know the committed colour before the roulette outcome is visible. To mitigate this risk, the casino may enforce the selection of a specific one-way function. However, the gambler may suspect that the suggested function has a backdoor known to the casino, allowing them to know the committed colour in advance and potentially influencing the outcome of the bet.

In other words, if the party committing to the value determines the function, they can intentionally choose a function with collisions for the committed value. On the other hand, if the verifier (once the committed value is revealed) selects the one-way function, it may have a backdoor that prematurely reveals the committed value.

Allowing both parties to select a one-way function to address these concerns can be advantageous. The approach could involve using the one-way function chosen by the party to whom the secret will be revealed first, thereby eliminating planned collisions by the committing party. After that, another one-way function chosen by the committing party can be applied to the result, thus avoiding the existence of a backdoor in the first one-way function.

Care should be taken when nesting hash functions to ensure low implied collisions. Maintaining a small number of collisions in each nesting stage is crucial overall. Sampling techniques similar to those employed in self-testing scenarios discussed in references, such as [BLR90] and [DF09] can be utilized to estimate the probability of collisions.

Additionally, nesting can be combined with using polynomials to eliminate potential collisions further, as presented in the previous section.

7 Concluding Remarks

Pursuing one-way functions that can be proven secure is closely interconnected with investigating a fundamental milestone in computer science known as the $P \neq NP$ problem. Employing information-theoretically secure primitives as building blocks, such as one-time pads, (pseudo) random permutations, and secret sharing, shows great potential to strengthen efficient cryptographic commitments. This paper presents several efficient commitment schemes that are not based on number theory (that may be attacked by quantum computers as DH and RSA are attacked) and are lightweight and, thus, can fit the Internet of Things scenarios. By incorporating error detection techniques and permutations, we have effectively reduced the incidence of collisions and eliminated linearity.

Finally, we introduce a novel and efficient candidate for a one-way function based on a Feistel network combined with the usage of polynomials to avoid collisions.

References

[BLR90] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, pages 73–83, 1990.

[BM89] László Babai and Shlomo Moran. Proving properties of interactive proofs by a generalized counting technique. *Information and Computation*, 82:185–197, 1989.

[CCD23] Asaf Cohen, Pawel Cyprys, and Shlomi Dolev. Single instance self-masking via permutations. IACR Cryptology ePrint Archive, 2023. Paper 416.

[CDM22] Pawel Cyprys, Shlomi Dolev, and Shlomo Moran. Self masking for hardening inversions, 2022. Paper 1274.

[CDM23] P. Cyprys, S. Dolev, and O. Margalit. Owf candidates, examples implementations, 2023. https://bit.ly/code-implementation.

[DD20] H. Dolev and S. Dolev. Toward provable one way functions, 2020. IACR Cryptology ePrint Archive Paper 1358.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34, 2021.

[DF09] Shlomi Dolev and Sivan Frenkel. Extending the scope of self-correcting. In *Proceedings of the XIII International Conference on Applied Stochastic Models and Data Analysis (ASMDA-2009)*, 2009.

[DLH13] Shlomi Dolev, Limor Lahiani, and Yinnon A. Haviv. Unique permutation hashing. *Theor. Comput. Sci.*, 475:59–65, 2013.

[FGT92] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Appl. Math.*, 39(3):207–229, nov 1992.

[FJM18] Marc Fischlin, Christian Janson, and Sogol Mazaheri. Backdoored hash functions: Immunizing hmac and hkdf. In *31st IEEE Computer Security Foundations Symposium, CSF 2018*, pages 105–118. IEEE, 2018.

[FS87a] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[FS87b] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO' 86*, pages 186–194, 1987.

[GLN11] Oded Goldreich, Leonid A. Levin, and Noam Nisan. *On constructing 1-1 one-way functions*, page 13–25. Springer-Verlag, Berlin, Heidelberg, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18, 1989.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[Han11] Helena Handschuh. Sha-0, sha-1, SHA-2 (secure hash algorithm). In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 1190–1193. Springer, 2011.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 28:12–24, 1999.

[IN96] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9:199–216, 1996.

[Knu02a] Lars R. Knudsen. The security of feistel ciphers with six rounds or less. *J. Cryptol.*, 15(3):207–222, 2002.

[Knu02b] Lars R. Knudsen. The security of feistel ciphers with six rounds or less. *J. Cryptology*, 15:207–222, 2002.

[Lam79] L. Lamport. Constructing digital signatures from a one way function, 1979.

[Leu08] Gaëtan Leurent. Md4 is not one-way. In *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 412–428. Springer, 2008.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[RS91] I. Reed and G. Solomon. Itu-t study group 15 frame alignment and cyclic redundancy check (crc) procedures relating to basic frame structures defined in recommendation g.704, 1991. International Telecommunication Union.

[SA09] Yu Sasaki and Kazumaro Aoki. Finding preimages in full md5 faster than exhaustive search. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.

[Sha49] C. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28:656–715, 1949.

[Sho94] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[SLDW07] Marc Stevens, Arjen Lenstra, and Benne De Weger. Chosen-prefix collisions for md5 and colliding x.509 certificates for different identities. In *Advances in Cryptology-EUROCRYPT 2007*, pages 1–22. Springer, 2007.

[The23] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.8.0)*, 2023. Accessed: 2023.

A The Insufficiency of Linear Error Detection Codes, Toy Example

We present a simple example of numerical values to the reader to show the risk of using linear error detection. The example is later used to introduce permutations to eliminate linearity.

Let us consider the following example. Define the binary vector $r = [10011010]$. Compute $f(s_1, s_2) = (s_1 \circ crc_1) \oplus (crc_2 \circ s_2) = [11010111]$. To reverse the process of this operation, define an identity matrix I with eight rows and eight columns. Let s_1^i and s_2^i be defined as the first four bits and the next four bits in the $i'th$ row of the matrix I. Define matrix A as follows: the i'th row of the matrix consists of $(s_1^i \circ crc_1^i) \oplus (crc_2^i \circ s_2^i)$, where s_1^i (s_2^i) are the four first (last, respectively) bits in the ith row of the identity matrix I and crc_j^i is the CRC result over s_j^i. Below is an example of numeric values to help the reader understand the process. We use r to denote the result of f (also called the Blackbox) over s_1, s_2, which are, in our toy example, four bits each.

$$s_1 \circ s_2 = \begin{bmatrix} 1\,0\,0\,1\,1\,0\,1\,0 \end{bmatrix} \; r = \begin{bmatrix} 1\,1\,0\,1\,0\,1\,1\,1 \end{bmatrix}$$

$$A = \begin{bmatrix} (s_1^1 \circ crc_1^1) \oplus (crc_2^1 \circ s_2^1) \\ (s_1^2 \circ crc_1^2) \oplus (crc_2^2 \circ s_2^2) \\ \vdots \\ (s_1^8 \circ crc_1^8) \oplus (crc_2^8 \circ s_2^8) \end{bmatrix}$$

$$A = \begin{bmatrix} 1\,0\,0\,0\,1\,0\,1\,0 \\ 0\,1\,0\,0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,1\,1\,1\,0 \\ 0\,0\,0\,1\,0\,1\,1\,1 \\ 1\,0\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,0\,1\,0\,0 \\ 0\,1\,1\,0\,0\,0\,1\,0 \\ 0\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix} \; A^{-1} = \begin{bmatrix} 0\,1\,1\,1\,1\,1\,0\,0 \\ 1\,1\,1\,1\,0\,1\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1\,1 \\ 1\,0\,0\,1\,1\,1\,0\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,1\,0\,1\,1\,1\,1 \\ 0\,0\,1\,1\,1\,0\,1\,1 \\ 1\,1\,0\,1\,1\,0\,0\,1 \end{bmatrix}$$

$$f(s_1, s_2) \cdot A^{-1} = \begin{bmatrix} 1\,1\,0\,1\,0\,1\,1\,1 \end{bmatrix} \begin{bmatrix} 0\,1\,1\,1\,1\,1\,0\,0 \\ 1\,1\,1\,1\,0\,1\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1\,1 \\ 1\,0\,0\,1\,1\,1\,0\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,1\,0\,1\,1\,1\,1 \\ 0\,0\,1\,1\,1\,0\,1\,1 \\ 1\,1\,0\,1\,1\,0\,0\,1 \end{bmatrix}$$

Which results in:

$$\begin{bmatrix}1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\end{bmatrix} = s_1 \circ s_2$$

We recovered the original values from the output of the Blackbox. All-time complexity (matrix multiplication and matrix inversion) is polynomial. The previously described operations are described in the following pseudocode:

The CRC-4 toy example.

Algorithm 1: Polynomial time inversion with CRC-4-ITU

```
Function BlackBox($s_1$, $s_2$):
    $z_1 = s_1 \circ calculate\_crc(s_1)$
    $z_2 = calculate\_crc(s_2) \circ s_2$
    return $z_1 \oplus z_2$
Function GenerateMatrix():
    A_matrix = Empty $0 \times 8$ matrix
    foreach row in (all 8 unit vectors) do
        x = BlackBox(row[0:3], row[4:8])
        A_matrix.add_to_matrix(row.index(), x)
    return A_matrix
Function Inverse():
    $r_{12} = BlackBox(s_1, s_2)$
    A_matrix = Generate_Matrix()
    return recorvered_$s_1 s_2 = r_{12} \times A\_matrix.inverse()$
```

Description of the pseudocode. The pseudocode commences by defining a function named “BlackBox.” This function takes two binary strings, denoted as s_1 and s_2, as input. Subsequently, the CRC-4 value is computed for each of these strings. The result of the function is a binary string obtained from performing the bitwise xor operation, represented as $z_1 \oplus z_2$. Following that, another function called “GenerateMatrix” is introduced. This function constructs a diagonal matrix with a size equivalent to the length of the binary string obtained from the previous xor operation. Within a loop, the “BlackBox” function is invoked to calculate the values for the first four elements (designated in the $[n : m]$ list notation, where n is the index of the first element and m is the index of the last element) of each row. Subsequently, the function calculates the values for the remaining four elements. These computed values are then used to create each matrix row, denoted as A_{matrix}. Finally, the function returns the resulting matrix. The last function, referred to as “Inverse,” is the program’s main function. It begins by calling the “BlackBox” function with the selected inputs s_1 and s_2, which produces a vector named r_{12}. Subsequently, this vector is utilized to restore the original values of s_1 and s_2 by performing multiplication with the inverted A_{matrix}.

B Polynomials for collision prevention

Assume a sufficiently large finite field F and $k+2$ distinct numbers in F, denoted as $x_1, \ldots, x_{k+2}$ (with the possibility of x_i being equal to i). The pseudocode for preventing polynomial collisions is presented in Algorithm 2. As input to the algorithm, we generate the commitment (line 1) and employ it as part of secret sharing, where the constant term a_0 of the polynomial $P(x) = a_1 x + a_0$ defined over the finite field F represents the committed value (in our example, the result of f using s_1 and s_2). These values are declared as inputs to Algorithm 2. Next, we generate a random value from F to encode the polynomial coefficient a_1 (line 3). Finally, we employ a popular algorithm - Lagrange Interpolation to construct the polynomial $f(x)$ (line 4) given points.

Algorithm 2: Polynomial generation

```
Input: $n = k + 2$ distinct numbers, $F = GaloisField(2^\ell)$, commitment
Result: $f(x)$ polynomial
1 Function Generate_Input():
2     $a_0 = commitment$
3     $a_1 = Generate\_Random\_Point(n, F)$
4     $f(x) = Lagrange\_Interpolation(a_0, a_1)$
```

Continuing from the previous section, consider a Finite Field with an order of 2^ℓ, where each element in the field consists of precisely ℓ bits. As mentioned above, let t be the input for Algorithm 3. The subsequent step involves evaluating the polynomial $P(x)$ for input values $x = x_1, x_2, \ldots, x_{k+2}$ to generate a vector $t_1, t_2, \ldots, t_{k+2}$ (refer to lines 2 to 4 in Algorithm 3).

Subsequently, we can utilize the binary representation of t_i (line 5) to encode two strings s_1^i and s_2^i (line 6). Specifically, we take the first half of $\ell/2$ bits from t_i to form s_1^i, and in the subsequent line, we take the other half of $\ell/2$ bits to form s_2^i. Following this, we can calculate the permuted hash for each string (line 8) in a way described in Section 3.

It's important to note that the resulting value of $f(s_1^i, s_2^i)$ may not be unique, as there may exist s_1' and s_2' for which $f(s_1', s_2') = f(s_1^i, s_2^i)$, indicating the presence of collisions.

To improve the resistance against inversion attacks, we can empower the verifying party in the commitment process by allowing them to choose multiple values of x. They can request the corresponding committed values (y) before revealing the next challenge value of x.

The following figures demonstrate the situation where the committer can expose one of several lines (and corresponding commitments) when only the values for $x = 1$ and $x = 2$ are requested (indicated by the green colour). Define $P(x) = a_1 x + a_0$, where a_0 represents the committed value (preimage of the one-way function), and a_1 is a randomly chosen value from the field. In Figure 1, the value $f(s_1, s_2)$ (or $f(s_3, s_4)$) is depicted as a blue horizontal line, where $s_1 \circ s_2$ corresponds to $a_0 + a_1$, and $s_3 \circ s_4$ corresponds to $a_0 + 2a_1$.

Algorithm 3: Calculation of permuted hash

```
  Input: $f(x) = polynomial$, $F = GaloisField(2^\ell)$
  Result: $h = permuted\ hash$
1 Function Generate_Values():
2     $t_1 = f(x_1)$
3     $\vdots$
4     $t_2 = f(x_{k+2})$
5     $t_{i_{binary}} = binary\_cast(t_{x_i})$
6     $s_1 = t_{i_{binary}}[0{:}\ell/2]$
7     $s_2 = t_{i_{binary}}[\ell/2{:}\ell]$
8     $h = Calculate\_Permuted\_Hash(s_1, s_2)$
```

Figure 2 illustrates the green line representing $P(x)$, which shows the y coordinate for $x = 1$ (or $x = 2$) based on $s_1 \circ s_2$ (or $s_3 \circ s_4$) values. Interestingly, the blue horizontal lines in Figure 1 reveal a collision for $s_3 \circ s_4$, indicating the existence of $s'_3 \circ s'_4$ for which $f(s_3, s_4) = f(s'_3, s'_4)$. This collision implies that the committer can expose an additional committed value, which differs from the value represented by the green line.

When the committer needs to reveal $f(s_5, s_6)$, where $(s_5 \circ s_6)$ corresponds to the value of $P(3)$, the collisions indicated by the new blue line in Figure 3 do not align with the red line represented by $s_1 \circ s_2$ and s'_3, s'_4 (see Figure 4). Consequently, the committer is compelled to reveal the points on the green line, effectively exposing the original committed value.

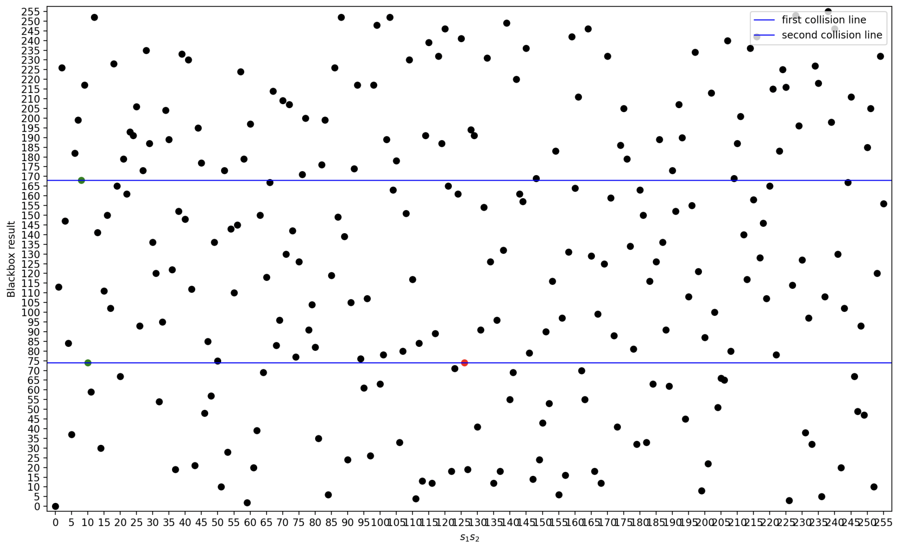

Fig. 1. Black-box values distribution for all possible s_1 and s_2 along with collision lines for $r_{1,2}$ and $r_{3,4}$

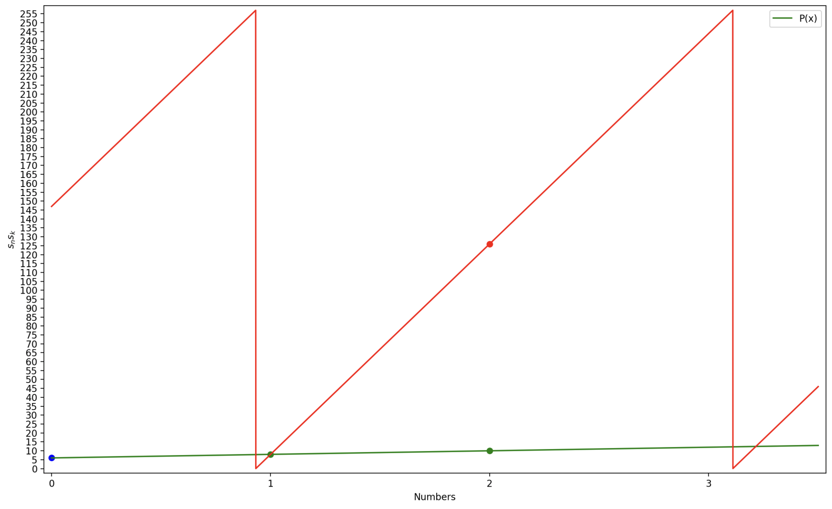

Fig. 2. Collision line for $P(2)$ points

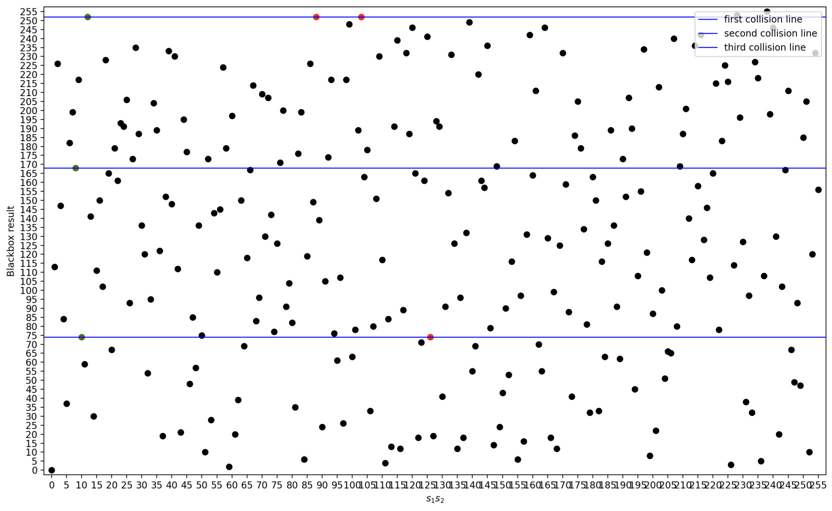

Fig. 3. Black-box values distribution for all possible s_1 and s_2 along with collision lines for $r_{1,2}$ and $r_{3,4}$

Note that such an approach can be relevant to other cryptographic hash functions where the input is padded by a random nonce chosen by the other party. The nonce can also encode a (partial) permutation index given to the committer.

Possibly, the verifier and the committer may agree that the committer will permute the message and concatenate it with the nonce. Interaction can be

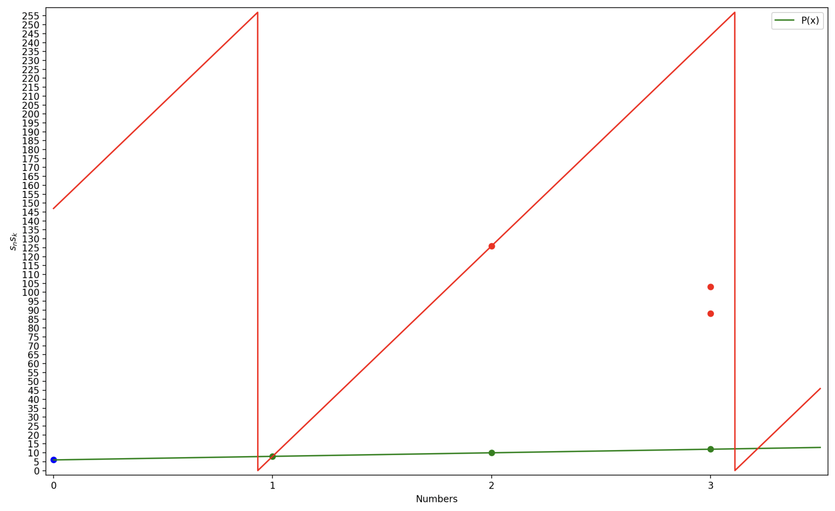

Fig. 4. Collision lines for $P(3)$ points

eliminated by the use of the Fiat-Shamir random oracle [FS87b], where the next x coordinate is a function (say, xor based) of (several or all) the y values obtained so far.

C MD5 collision example

MD5 serves as a cryptographic hash function designed to verify data integrity by generating a fixed-size hash value from variable-length input data. However, its susceptibility to collision vulnerabilities undermines its reliability. Specifically, MD5 can erroneously produce identical hash outputs for distinct inputs, for example:

First Pair:

– **Sequence 1A:**

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab58712467eab 4004583eb8fb7f89
55ad340609f4b302 83e488832571415a 085125e8f7cdc99f d91dbdf280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e2b487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080a80d1e c69821bcb6a88393 96f9652b6ff72a70
```

– **Sequence 1B:**

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab50712467eab 4004583eb8fb7f89
55ad340609f4b302 83e4888325f1415a 085125e8f7cdc99f d91dbd7280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e23487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080280d1e c69821bcb6a88393 96f965ab6ff72a70
```

Second Pair:

– **Sequence 2A:**

```
4dc968ff0ee35c20 9572d4777b721587 d36fa7b21bdc56b7 4a3dc0783e7b9518
afbfa200a8284bf3 6e8e4b55b35f4275 93d849676da0d155 5d8360fb5f07fea2
```

– **Sequence 2B:**
`4dc968ff0ee35c20 9572d4777b721587 d36fa7b21bdc56b7 4a3dc0783e7b9518 afbfa202a8284bf3 6e8e4b55b35f4275 93d849676da0d1d5 5d8360fb5f07fea2`

C.1 Collision Sample Pairs

Choose a line out of the four defined by the two first pairs (for two pre-agreed fixed x values), then use the y on this line that corresponds to the x (on this line) where, say, the xor of the previous two defines the x hashed y's (in the spirit of Fiat Shamir usage of hash results). Given MD5's vulnerability to chosen prefix collisions, we identify two pairs of sequences. Despite their differences, each pair results in the same MD5 hash.

Pair 1: Sequence 1A and Sequence 1B differ slightly but share the same MD5 hash:

`79054025255fb1a26e4bc422aef54eb4`.

The variation is minimal, typically in a few hexadecimal characters, illustrating a chosen-prefix collision.

Pair 2: Similarly, Sequence 2A and Sequence 2B have minor differences but result in the identical MD5 hash:

`82aee5d7c2f182c2d34d2e48d0d9089e`,

Pair 3: Similar to the previous examples, Sequence 3A and Sequence 3B contain slight alterations yet produce the same MD5 hash:

`cee9a457e790cf20d4bdaa6d69f01e41`,

Each sequence within its pair confirms MD5's collision issue. The commitment is represented by one of these sequences. However, without further differentiation, which pair accurately represents the commitment remains unclear, highlighting MD5's critical flaw.

D Polynomial MD5 collisions

Figures 5 and 6 demonstrate a new method for securing digital commitments, aiming to overcome the weaknesses of the MD5 (or any other) hash function, which is known to be vulnerable to attacks where different inputs (also may occur when the output of MD5 is of the length of the input, and therefore when the pigeonhole consideration cannot be applied) produce the same output, known as collisions. We can apply this idea to any other commitment scheme to deal with the potential of undesired collisions.

D.1 Details in the Figures

The figures illustrate the line intersecting the y-axis at the point of commitment. To commit to b, one randomly chooses a to form $P(x) = ax+b$. Then, it computes $y = P(x)$ for each agreed-upon x and calculates and outputs the MD5 hash of y. Once two such hash function results are public, in the example in the figures, results of MD5($P(2)$) and MD5($P(4)$) computations of MD5($P(x)$) for more x's are required to be outputted, MD5($P(7)$) in the case illustrated in Figure 6. Note that, to avoid the need for interaction, the commitment agreement can be to output MD5($P(1)$, MD5($P(2)$) and then compute the next x (in the spirit of Fiat-Shamir technique) by, say, x_3=MD5($P(1)$) xor MD5($P(2)$), outputting MD5($P(x_3)$), similarly the next x, x_4 can be a function of all previous hash results. If x_i already participated, the first ("clockwise" order) subsequent non-participating x is used.

The figures show two points for each chosen x-value. These pairs of points represent MD5 collisions, which are sequences of data that result in the same MD5 hash, namely, the points labelled M_{1A} and M_{1B} in the figures correspond to different inputs but lead to the same MD5 hash, depicted by a red dashed line. This is also true for the points labeled M_{2A} and M_{2B}.

If a person knows that their hash function is prone to collisions, they can reduce the chance of dishonest behaviour by carefully choosing the next x value. Upon selecting this x, they evaluate the polynomial at that point and then use this result to compute the MD5 hash. This process is illustrated in Figure 6. We theorize that as more x values are selected in this manner, it becomes increasingly difficult for a potential attacker to discover collisions that match the original polynomial. We depict them as points with an "Unlikely Collision" label. Consequently, we assert that the likelihood of successfully manipulating the system exponentially decreases with each additional x chosen.

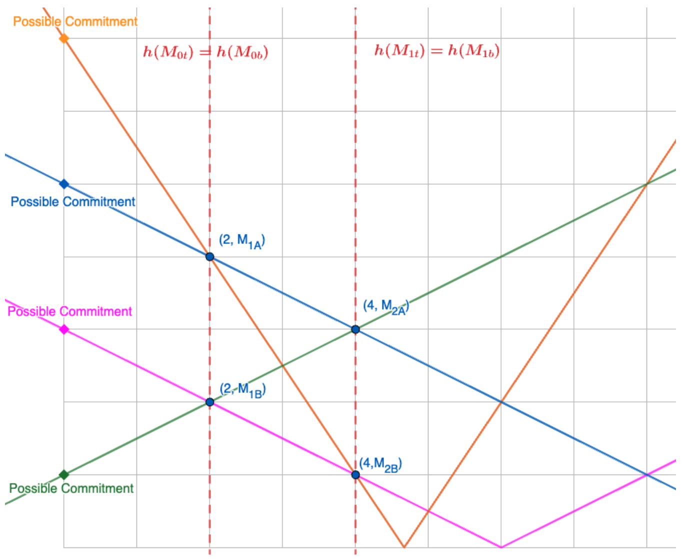

Fig. 5. Polynomial Commitment Intersection and MD5 Hash Evaluation

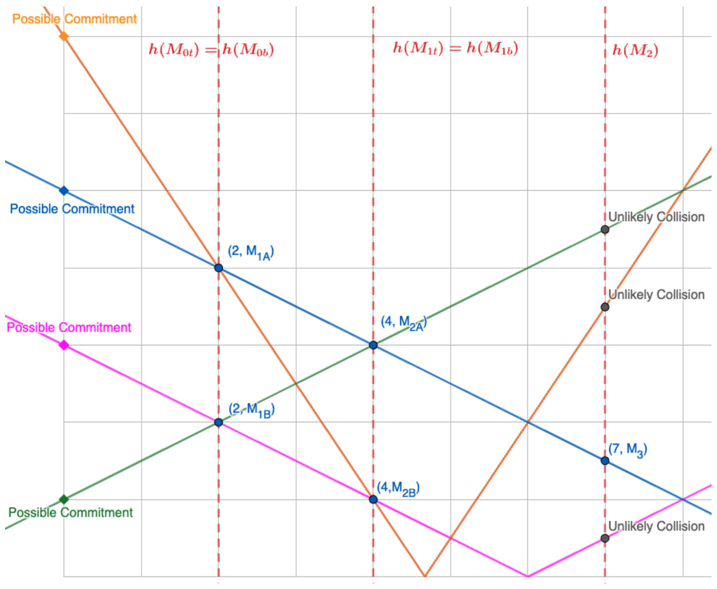

Fig. 6. Extended Polynomial Commitment Intersection and MD5 Hash Evaluation