Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers

Phillip Rogaway[1] and John Steinberger[2]

[1] Department of Computer Science, University of California, Davis, USA
[2] Department of Mathematics, University of British Columbia, Canada

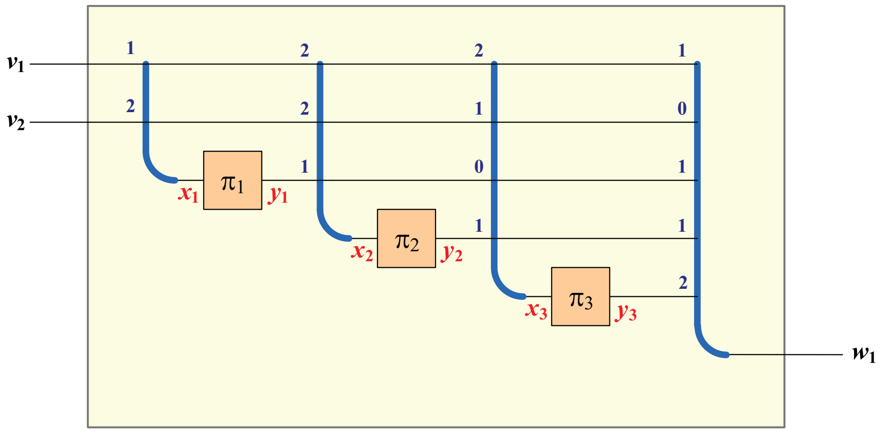
Abstract. We propose a family of compression functions built from fixed-key blockciphers and investigate their collision and preimage security in the ideal-cipher model. The constructions have security approaching and in many cases equaling the security upper bounds found in previous work of the authors [24]. In particular, we describe a $2n$-bit to n-bit compression function using three n-bit permutation calls that has collision security $N^{0.5}$, where $N = 2^n$, and we describe $3n$-bit to $2n$-bit compression functions using five and six permutation calls and having collision security of at least $N^{0.55}$ and $N^{0.63}$.

Key words: blockcipher-based hashing, collision-resistant hashing, compression functions, cryptographic hash functions, ideal-cipher model.

1 Introduction

This paper is about fixed-key constructions for turning blockciphers into compression functions. When we say that a blockcipher-based construction is *fixed key* we mean that just a handful of constants are used as keys, so that our starting point is actually a small collection of permutations. The idea of doing cryptographic hashing from such a starting point was introduced by Preneel, Govaerts, and Vandewalle some 15 years ago [20], but the approach did not catch on.

For years, the customary starting point for building cryptographic hash functions has been (non-fixed-key) blockciphers, even if this hasn't always been made explicit. But a fixed-key design has some definite advantages. For one thing, blockciphers usually have significant key-setup costs, so fixing the keys can be good for efficiency. More than that, blockcipher designs are typically not very conservative with respect to related-key attacks—but this and more is needed when a blockcipher is used in any standard way to build a collision-resistant hash function. A fixed-key design effectively addresses this concern, banishing the need for related-key-attack security on a large space of keys. Finally, a fixed-key design concentrates the cryptographic work into a highly compact primitive, say a map π: $\{0,1\}^{128} \to \{0,1\}^{128}$ instead of a map E: $\{0,1\}^{512} \times \{0,1\}^{160} \to \{0,1\}^{160}$. Cryptographically processing fewer than half as many bits, a fixed-key design embodies an aspiration for minimalism.



```
algorithm LP$^A_{231}$ ($v_1\,v_2$)
$x_1 \leftarrow v_1 + \mathbf{2}v_2$
$y_1 \leftarrow \pi_1(x_1)$
$x_2 \leftarrow \mathbf{2}\,v_1 + \mathbf{2}\,v_2 + y_1$
$y_2 \leftarrow \pi_2(x_2)$
$x_3 \leftarrow \mathbf{2}\,v_1 + v_2 + y_2$
$y_3 \leftarrow \pi_3(x_2)$
$w_1 \leftarrow v_1 + y_1 + y_2 + \mathbf{2}\,y_3$
return $w_1$
```

```
algorithm LP$^A_{mkr}$ ($v_1 \cdots v_m$)
for $i \leftarrow 1$ to $k$ do
    $x_i \leftarrow a_i \cdot (v_1, \ldots, v_m,\ y_1, \ldots, y_{i-1})$
    $y_i \leftarrow \pi_i(x_i)$
for $i \leftarrow 1$ to $r$ do
    $w_i \leftarrow a_{k+i} \cdot (v_1, \ldots, v_m,\ y_1, \ldots, y_k)$
return $w_1 \cdots w_r$
```

Fig. 1. Top & bottom left: Compression function LP$^A_{231}$ (usually denoted LP231) for a suitable matrix $A = (a_{ij})$. Horizontal lines carry $n{=}128$ bit strings and x_1, x_2, x_3, w_1 are computed as in $x_3 = \mathbf{2}\,v_1 + v_2 + y_2$ with arithmetic and constants $\mathbf{0}$, $\mathbf{1}$, and $\mathbf{2} = 0^{126}10$ all in $\mathbb{F}_{2^{128}}$. Permutations π_1, π_2, π_3: $\{0,1\}^n \to \{0,1\}^n$ are modeled as independent random permutations to which the adversary can make forward and backwards queries. **Bottom right:** Defining LP$^A_{mkr}$ for an $(k{+}r) \times (k{+}m)$ matrix A over $\mathbb{F}_{2^n}$. Function lp$^A_{mkr}$ is identical but uses a single permutation π.

So far, it has not been clear that it is *possible* to turn an n-bit permutation into a hash function that outputs n or more bits and has desirable collision-resistance bounds. Nobody has ever demonstrated such a design, and, three years ago, Black, Cochran, and Shrimpton seemed to cast a shadow on the possibility [6]. They showed that a prior construction in the literature was wrong, in the sense of having a query-efficient attack, and that, in fact, so will *any* iterated hash function whose underlying compression function maps $2n$ bits to n bits using a single call to an n-bit permutation. But Black *et al.* never said that provably-sound permutation-based hashing is impossible—only that such a scheme couldn't be *extremely* efficient.

In earlier work, the authors extended the Black *et al.* findings [24]. We showed that one expects to find a collision in an $m \xrightarrow{k} r$ compression function—meaning one that maps mn bits to rn bits by calling a sequence of k n-bit permutations—with about $N^{1-(m-0.5r)/k}$ queries, where $N = 2^n$. The result is in the ideal-cipher model and assumes a random-looking compression-function, formalized by

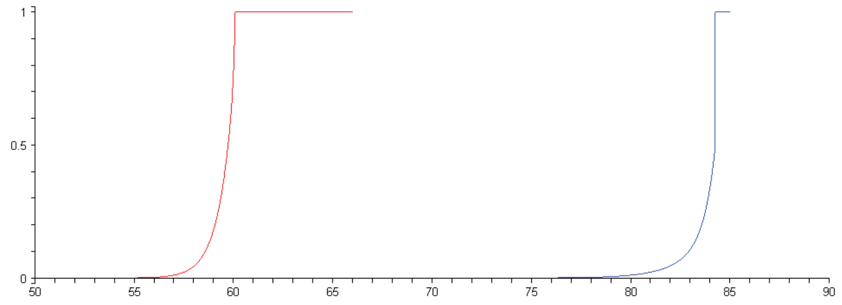


Fig. 2. Proven collision resistance (left curve) and preimage resistance (right curve) for LP231 with $n = 128$ and a suitable matrix A, as given by Theorems 1 and 2. In the ideal-permutation model, no adversary asking $q = 2^x$ queries (the x-axis) can have find a collision (left) or preimage (right) with probability exceeding the value shown.

a notion of *collision-uniformity*. The result suggests that a desirable-in-practice $2 \xrightarrow{2} 1$ construction can't deliver acceptable security so a $2 \xrightarrow{3} 1$ design is the best one can hope for in a good single-length construction. Similarly, a collision-uniform $3 \xrightarrow{4} 2$ construction can't have a satisfying provably-good bound, and even a (collision-uniform) $3 \xrightarrow{5} 2$ construction will fail at around $N^{3/5}$ queries, and a $3 \xrightarrow{6} 2$ design at around $N^{2/3}$ queries. The same paper also proves that the preimage resistance of a $m \xrightarrow{k} r$ scheme is limited to about $N^{1-(m-r)/k}$ queries, again assuming random-like behavior, now formalized as *preimage uniformity*. Stam has recently shown that the collision-uniformity assumption cannot be removed [29].

Results. In this paper we give practical constructions that approach the limits described above for uniform permutation-based compression functions. Given numbers n, m, k, and r, and given an appropriate matrix A, we define an $m \xrightarrow{k} r$ compression function LP^A_{mkr}. The matrix A is $(k+r) \times (k+m)$ entries of n-bit strings, which we regard as points in the field $\mathbb{F}_{2^n}$. We will often omit mention of A and, when we do, move the subscripts up, say writing LP231 in lieu of LP^A_{231}. By varying A the LP^A_{mkr} schemes encompass *all* permutation-based $m \xrightarrow{k} r$ constructions where each permutation's input as well as each n-bit block of output is linearly determined from all that's come before. See Fig. 1. We call a function from this schema an LP compression-function (linearly-determined, permutation-based).

We first study LP231, the smallest potentially "good" LP scheme. We exhibit a condition on the matrix A for which the scheme demonstrably achieves security to about $N^{1/2}$ queries, which is of course optimal for any hash function that outputs n bits. Our analysis is in terms of concrete security, and for $n = 128$, it turns out that the adversary must ask more than $2^{59.72}$ queries to get a 0.5 chance to find a collision. See the left-hand curve of Fig. 2.

Besides collision resistance we also consider the preimage resistance of LP231. Under the same conditions on its matrix, the scheme has asymptotic security of $N^{2/3}$ queries, which is optimal for any preimage-uniform $2 \xrightarrow{3} 1$ scheme [24].

scheme	maps	collision resistance			preimage resistance		
		security	attack	tight?	security	attack	tight?
LP231, lp231	$2 \xrightarrow{3} 1$	$N^{0.50}$	$N^{0.50}$	✓	$N^{0.67}$	$N^{0.67}$	✓
LP241, lp241	$2 \xrightarrow{4} 1$	$N^{0.50}$	$N^{0.50}$	✓	$N^{0.75}$	$N^{0.75}$	✓
LP352, lp352	$3 \xrightarrow{5} 2$	$N^{0.55}$	$N^{0.60}$		$N^{0.80}$	$N^{0.80}$	✓
LP362, lp362	$3 \xrightarrow{6} 2$	$N^{0.63}$	$N^{0.66}$		$N^{0.80}$	$N^{0.83}$	
$\mathrm{LP}^{\mathrm{SS}}_{231}$	$2 \xrightarrow{3} 1$	$N^{0.50}$	$N^{0.50}$	✓	$N^{0.50}$	$N^{0.67}$	
$\mathrm{lp}^{\mathrm{SS}}_{231}$	$2 \xrightarrow{3} 1$	N^{0}	N^{0}		N^{0}	N^{0}	

Fig. 3. Rows 1–4: Automated analyses of our schemes instantiated with an appropriate sequence of matrices. The attacks are from prior work [24]. **Rows 5–6:** Automated analysis of the SS-scheme [28] and its single-permutation variant.

Numerically, for $n = 128$ one must ask more than $2^{84.25}$ queries to get a 0.5 chance to find a given preimage. See the right curve of Fig. 2.

Next we look at LP352 and LP362. Such double-length schemes are important because, in practice, the source of our permutations is likely to be AES, and its blocklength of $n = 128$ is below what people want in a hash function's output. We also look at the LP scheme just like LP^{A}_{mkr} except that a single permutation is used throughout, instead of k independent ones. This is desirable because it yields a more minimalist scheme, one needing less hardware or a smaller memory footprint. Let lp^{A}_{mkr} be defined as in Fig. 1 but with all permutations the same. When we don't want to specify A we write lp231, lp352, and so on.

For the mechanisms named in the last paragraph, doing an analysis like we did for LP231 seems infeasible, generating too many cases. We therefore developed a computer program that is capable of carrying out these analyses. It is fed a matrix A and computes asymptotic upper bounds for collision resistance and preimage resistance. For LP352 our program establishes collision resistance to at least $N^{0.55}$ queries; for LP362, at least $N^{0.63}$. The same program finds bounds for the single-permutation constructions lp231, lp352, and lp362 that are the same as for the corresponding multiple-permutation schemes. See Fig. 3. All of these results assume an appropriate matrix A (see Section 4).

While collision security of $N^{0.55}$ or $N^{0.63}$ may not seem excessive for a compression function of output length $2n$ (whose collision security could ideally reach N), one should bear in mind that the attacks described in [24] assume information-theoretic adversaries. We do not know any *actual* (implementable) attacks for LP352 or LP362 that have running time less than N. Also, collision and preimage security may increase when compression functions are used in an iterated construction.

Further related work. Shrimpton and Stam describe a $2 \xrightarrow{3} 1$ compression function based on an n-bit to n-bit random *function* [28]. While cryptographic practice does not seem to directly provide such objects, their construction is the first to obtain a good collision-resistance bound starting from a non-compressing primitive. To make a permutation-based scheme they suggest to instantiate each random function as a permutation with a feed-forward xor. They go on to explain that it's equivalent to modify just the first two, resulting in a scheme, call it $\mathrm{LP}^{\mathrm{SS}}_{231}$, that is LP^{A}_{231} where A has rows 10000, 01000, 11110, and 10101. This matrix does not satisfy the criterion associated to our concrete security bound, but our computer-based technique can handle it, proving asymptotic collision-resistance of $N^{0.5}$ queries. Preimage resistance is also $N^{0.5}$, shy of the desired $N^{2/3}$. This is not a deficiency in the analysis, as it matches a known attack by Joux [28]. We point out that $\mathrm{lp}^{\mathrm{SS}}_{231}$, the single-permutation variant of $\mathrm{LP}^{\mathrm{SS}}_{231}$, is completely insecure, admitting a two-query preimage-finding attack and a our-query collision-finding one. See Fig. 3.

An interesting recent hash function is the *sponge construction* of Bertoni, Daemen, Peeters, and Van Assche [4, 5]. The mechanism turns an n-bit permutation (or function) into an arbitrary-output-length hash function that is indifferentiable from a random oracle [8] (which is stronger than collision and preimage resistance). But the concrete security bounds shown do not enable its use with a 128-bit permutation; security is never as high as $N^{1/2}$, and approaching that is paid for with the scheme's rate.

2 Preliminaries

The model. A *compression function* is a map $H\colon \{0,1\}^{mn} \to \{0,1\}^{rn}$ for a given n, m, and r, with $m > r$. It is *permutation-based* if it is given by a (deterministic) program with oracle access to permutations $\pi_1, \pi_2, \ldots, \pi_k\colon \{0,1\}^n \to \{0,1\}^n$. Numbers k and n are constants associated to H. An *adversary* $\mathcal{A}$ for finding collisions in H is a probabilistic Turing machine with oracle access to the permutations used by H and their inverses. When $\mathcal{A}$ asks $(+1, i, x)$ it receives $y = \pi_i(x)$ (a *forward query*), and when it asks $(-1, i, y)$ it gets $x = \pi_i^{-1}(y)$ (a *backwards query*). We assume $\mathcal{A}$ never asks a *pointless* query, defined as a repeated query, asking $\pi_i(x)$ and later $\pi_i^{-1}(\pi_i(x))$, or asking $\pi_i^{-1}(y)$ and later $\pi_i(\pi_i^{-1}(y))$.

Now run the adversary $\mathcal{A}$ with its collection of oracles, instantiating each permutation by a uniformly chosen one. By the convention just given, the i-th query made by the adversary is answered by a uniform point from a set of size at least $N' = N - i + 1 > N - q$ where $N = 2^n$ and q is the total number of queries asked. Record the queries the adversary makes into a *query history* $\mathcal{Q}$ where the elements of $\mathcal{Q}$ are triplets (i, x, y). A triplet (i, x, y) is in the query history if the adversary asks $\pi_i(x)$ and gets back y, or it asks $\pi_i^{-1}(y)$ and gets back x. We usually specify the value of i in a triplet implicitly by way of the names of the other two arguments, like (x_1, y_1) for $(1, x_1, y_1)$, or (x_i', y_i') for (i, x_i', y_i').

The adversary *wins* if it outputs values v, v' with $v \neq v'$ such that $H(v) = H(v')$. We assume that the adversary makes the queries necessary to compute $H(v)$ and $H(v')$. As a result, one can tell from looking at the adversary's query history $\mathcal{Q}$ whether it has the information necessary to construct a collision. We thus dispense with the adversary having to output anything, giving the adversary the win if its query history contains a queries from which a collision in H can be constructed. Let $\mathbf{Adv}^{\mathrm{coll}}_{H}(\mathcal{A})$ be the probability that A wins. The probability is taken over the random permutations $\pi_1, \ldots, \pi_k$ and $\mathcal{A}$'s coins (if any). Let

$\mathbf{Adv}_H^{\mathrm{coll}}(q)$ be the maximal value of $\mathbf{Adv}_H^{\mathrm{coll}}(q)$ over all adversaries $\mathcal{A}$ that ask at most q queries, in all, to its oracles. Let $\mathsf{Coll}(\mathcal{Q})$ be the event that a collision can be constructed from queries in $\mathcal{Q}$. Note that $\mathbf{Adv}_H^{\mathrm{coll}}(\mathcal{A}) = \Pr[\mathsf{Coll}(\mathcal{Q})]$.

We can similarly define preimage resistance. The adversary's collection of oracles and the rules under which it operates are the same as for collision resistance, but now the adversary is said to win if it finds the preimage for a particular point $w \in \{0,1\}^{rn}$. Rather than asking the adversary to output the actual preimage, we can again look at the adversary's query history $\mathcal{Q}$ to see if it holds the information necessary to determine a preimage for w under H. We denote this predicate $\mathsf{Preim}_{H,w}(\mathcal{Q})$. We will usually omit the subscripts. We let $\mathbf{Adv}_H^{\mathrm{pre}}(\mathcal{A}) = \max_w\{\Pr[\mathcal{A}$ asks queries $\mathcal{Q}$ for which $\mathsf{Preim}_{H,w}(\mathcal{Q})]\}$. Informally, a compression function is preimage resistant if the adversary can't invert a given point without asking an unreasonable number of queries.

The subject schemes. Identify n-bit strings and points in the finite field $\mathbb{F}_{2^n}$. In this way n-bit strings inherit addition and multiplication. Given vectors of n-bit strings $x = (x_1, \ldots, x_a) \in (\{0,1\}^n)^a$ and $y = (y_1, \ldots, y_b) \in (\{0,1\}^n)^b$ let $x \cdot y$ be the n-bit string $\sum_{i=1}^{\min(a,b)} x_i y_i$ that is the inner product of the shorter vector and the truncated longer one. The i-th row of a matrix A is the vector a_i and its row-i, column-j element is a_{ij}.

Fix positive integers n, k, m and r and a $(k+r) \times (m+k)$ matrix A of n-bit strings and permutations $\pi_1, \ldots, \pi_k\colon \{0,1\}^n \to \{0,1\}^n$. We call these variables the *parameters* of our scheme. They induce a hash function $\mathrm{LP}_{mkr}^A\colon \{0,1\}^{mn} \to \{0,1\}^{rn}$ that is defined in Fig. 1. The number n is called the *blocksize*; m is the *input length*; r is the *output length*; k is the *number of permutation calls*; A is the scheme's *matrix*, and $(\pi_1, \ldots, \pi_k)$ are its *permutations*. If they are all different then we're in the *multiple-permutation* setting and the adversary has forwards and backwards access to each. If a single permutation π is used throughout then we're in the *single-permutation* setting and the adversary has forwards and backwards access to it. Compression functions LP_{mkr}^A (multiple permutations) and lp_{mkr}^A (a single permutation) differ only in this way.

3 Concrete Security Bounds for LP231

Our theorem statements refer to a function $\beta(q,p,b,B)$. It is defined as the probability that the sum of q independent random variables that are b with probability p and value 0 with probability $1-p$ exceeds B. It is easy to get good upper bounds on β. For example if $B < b$ then $\beta(q,p,b,B) \le pq$, and if $b \ne 0$ then, letting $x = B/b$ and $t = \lfloor x \rfloor + 1$, observe that, by the sum bound,

$$\beta(q,p,b,B) = \beta(q,p,1,x) \ \le\ p^t \binom{q}{t} . \tag{1}$$

This "binomial bound" is sharp enough for our purposes except for code associated to computing Corollary 4. There we bound β by combining the binomial bound and a standard Chernoff bound, selecting the smallest upper bound.

We will prove that LP231 achieves good collision resistance and preimage resistance if its matrix A satisfies a certain "independence criterion" that will be defined within the proof. A random matrix will satisfy the criterion with high probability, while a sample small-entry matrix A that works is

$$A = \begin{bmatrix} \mathbf{1}\,\mathbf{2}\,\mathbf{0}\,\mathbf{0}\,\mathbf{0} \\ \mathbf{2}\,\mathbf{2}\,\mathbf{1}\,\mathbf{0}\,\mathbf{0} \\ \mathbf{2}\,\mathbf{1}\,\mathbf{0}\,\mathbf{1}\,\mathbf{0} \\ \mathbf{1}\,\mathbf{0}\,\mathbf{1}\,\mathbf{1}\,\mathbf{2} \end{bmatrix} . \tag{2}$$

The numbers represent points in $\mathbb{F}_{2^{128}}$ by identifying their binary representation with coefficient vectors of a polynomial (eg, $\mathbf{3} = \mathtt{x}+1$). We use $\mathtt{x}^{128}+\mathtt{x}^7+\mathtt{x}^2+\mathtt{x}+1$ as our irreducible polynomial. We are now ready to state our main result on the security of LP231.

Theorem 1. *Fix $n, q \geq 1$ and let $H = \mathrm{LP}_{231}^{A}$ where $A \in \mathbb{F}_{2^n}^{4\times 5}$ satisfies the independence criterion. Let $N = 2^n$ and $N' = N - q$. Then, for any positive integers b_1, b_2, B_1, and B_2,*

$$\begin{aligned}\mathbf{Adv}_H^{\mathrm{coll}}(q) \leq\ & 12N\,\beta(q, 1/N', 1, b_1) + 4N\,\beta(q, 1/N', 1, b_2) \\ & +12N\,\beta(q, q/N', b_1, B_1) + 2N\,\beta(q, q/N', b_2, B_2) \\ & +4N\,\beta(q, q/N', b_1, B_2) + 3\,\beta(q, qB_1/N', 1, 0) + \beta(q, qB_2^2/N', 1, 0) .\end{aligned}$$

The content of Theorem 1 is a bit opaque, both because of the non-standard function β and the universally quantified b_1, b_2, B_1, B_2. Indeed for a given q and n one must optimize the constants b_1, b_2, B_1, B_2, likely by computer, to get the best possible bound. To clarify the strength of Theorem 1, the following two corollaries will help.

Corollary 1. *Let $H_n = \mathrm{LP}_{231}^{A_n}$ where $A_n \in \mathbb{F}_{2^n}^{4\times 5}$ satisfies the independence criterion. Let $\epsilon > 0$. Then $\lim_{n\to\infty} \mathbf{Adv}_{H_n}^{\mathrm{coll}}(2^{n/2-\epsilon}) = 0$.* $\square$

Corollary 2. *Let $H = \mathrm{LP}_{231}^{A}$ where $A \in \mathbb{F}_{2^{128}}^{4\times 5}$ satisfies the independence criterion. Then $\mathbf{Adv}_H^{\mathrm{coll}}(2^{59.72}) < 0.5$.* $\square$

The first corollary captures the asymptotic behavior of the formula in Theorem 1. The proof is in the full version of this paper [23]. There one chooses reasonable but non-optimal values of b_1, b_2, B_1, B_2 and the bound falls out. As commented on earlier, the asymptotic statement is the best one can hope for in a $2 \to 1$ construction because of the always-present birthday attack. The second corollary is obtained by computer-aided optimization of b_1, b_2, B_1, B_2 and q for $n = 128$. The selected constants are $(b_1, b_2, B_1, B_2) = (1, 1, 12, 12)$. In Fig. 2 we show a graph of our security bound for the case of $n = 128$ (the left-hand curve) with the choice of constants just named. The birthday attack (elided for clarity) would appear just to the right of that curve, rising from 0 to 1 with a midpoint at $q = 2^{65.7}$. The space between such curves is the "gap" in the proven collision-resistance of LP231. It is about a factor of 60.

For preimage resistance we have analogous results, with the bound pushed well to the right. The proof of the following is in the full version of this paper [23].

Theorem 2. *Fix $n, q \geq 1$ and let $H_n = \mathrm{LP}^{A}_{231}$ where matrix $A \in \mathbb{F}^{4\times 5}_{2^n}$ satisfies the independence criterion. Let $N = 2^n$ and $N' = N - q$. Then, for any positive integers b_1, b_2 and B_2,*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{pre}}_{H}(q) \leq 12N\,\beta(q, 1/N', 1, b_1) \,+\, 4N\,\beta(q, 1/N', 1, b_2) \\ +\, 2N\,\beta(q, q/N', b_2, B_2) \,+\, 4N\,\beta(q, q/N', b_1, B_2) \\ +\, \beta(q, B_2/N', 1, 0)\;. \quad \square\end{aligned}$$

We once again clarify the strength of the theorem through a couple of corollaries.

Corollary 3. *Let $H_n = \mathrm{LP}^{A_n}_{231}$ where $A_n \in \mathbb{F}^{4\times 5}_{2^n}$ satisfies the independence criterion. Let $\epsilon > 0$. Then $\lim_{n\to\infty} \mathbf{Adv}^{\mathrm{pre}}_{H_n}(2^{2n/3\,-\,\epsilon}) = 0$.* $\square$

Corollary 4. *Let $H = \mathrm{LP}^{A_n}_{231}$ where $A \in \mathbb{F}^{4\times 5}_{2^{128}}$ satisfies the independence criterion. Then $\mathbf{Adv}^{\mathrm{pre}}_{H}(2^{84.25}) < 0.5$.* $\square$

The proof of Corollary 3 is in the full version of this paper [23]. For Corollary 4, we select $b_1 = b_2 = 2$ and $B_2 = 2^{41.51}$, determined by computer optimization. In Fig. 2 we illustrate the preimage-resistance security bound for $n = 128$ as the right-hand curve.

While collision-resistance security is asymptotically as good as anything that can be achieved by an n-bit output compression function, preimage resistance is limited by the fact that this is a $2 \xrightarrow{3} 1$ schemes [24]. For $n = 128$, the known attack has at least a 0.5 chance of success with $2^{86.92}$ queries. This implies a "gap" in the proven preimage-resistance of LP231 of about a factor of 6.

Overview of the proof of Theorem 1. The proofs of the two theorems are similar. We will give a *very* high-level sketch of Theorem 1; due to space limitation, for the actual proof, we must direct the reader to the full version of this paper [23].

The problem of finding a collision is first reformulated in terms of solving a set of linear equations using variables from the query history. To upper bound the probability of the adversary obtaining a solution to the equations, we first upper bound the probability adversary getting lucky in certain ways that could be helpful in finding a solution, and then upper bound the probability of obtaining a solution assuming the adversary has not been "lucky". One obtains a final bound by adding the probability of getting lucky to the probability of obtaining a solution without being lucky. (Being "lucky" may mean, for example, finding a large number of solutions to a certain subsystem of equations.) In terms of events, one defines an event $\mathsf{Lucky}(\mathcal{Q})$ and then one gives separate upper bounds for $\Pr[\mathsf{Lucky}(\mathcal{Q})]$ and $\Pr[\neg\mathsf{Lucky}(\mathcal{Q}) \wedge \mathsf{Coll}(\mathcal{Q})]$; the sum of the two bounds is an upper bound for $\Pr[\mathsf{Coll}(\mathcal{Q})]$. In fact there are really two levels or "stages" of luckiness: one defines events $\mathsf{Lucky1}(\mathcal{Q})$ and $\mathsf{Lucky2}(\mathcal{Q})$, and then upper bounds the three probabilities $\Pr[\mathsf{Lucky1}(\mathcal{Q})]$, $\Pr[\neg\mathsf{Lucky1}(\mathcal{Q}) \wedge \mathsf{Lucky2}(\mathcal{Q})]$ and $\Pr[\neg\mathsf{Lucky2}(\mathcal{Q}) \wedge \mathsf{Coll}(\mathcal{Q})]$, and takes the sum of the three terms. These three terms correspond to the three lines in the bound of Theorem 1 (in that order). The event $\mathsf{Lucky1}$ is defined in terms of parameters b_1 and b_2 and the event $\mathsf{Lucky2}$ is defined in terms of parameters B_1 and B_2, hence the appearance of these variables in the statement of Theorem 1.

The above description is still rather simplified; for example event $\mathsf{Lucky1}(\mathcal{Q})$ is really the disjunction of two events, "$\mathsf{WinA}(\mathcal{Q})$" and "$\mathsf{WinB}(\mathcal{Q})$", which are themselves the disjunction of 12 and 4 different events. Similarly $\mathsf{Lucky2}(\mathcal{Q})$ is the disjunction of three events—"$\mathsf{WinC}(\mathcal{Q})$", "$\mathsf{WinD}(\mathcal{Q})$" and "$\mathsf{WinE}(\mathcal{Q})$"—which are themselves the disjunction of 12, 2 and 4 different events each. Finally $\mathsf{Coll}(\mathcal{Q})$ is shown equivalent to an event $\mathsf{WinF}(\mathcal{Q})$ which is the disjunction of 4 different events. All in all, approximately 40 different named events are considered. This kind of multilevel decomposition of the collision event into sub-events is similar to the technique employed in the analysis of MDC-2 [30].

4 Automated Analyses of LP Compression Functions

Overview. We now describe the theory underlying a computer program we designed to get asymptotic security bounds like those of Corollaries 1 and 3 but for constructions larger than LP231, where the analysis gets too complex to do by hand. The method is applicable both to collision-resistance and preimage-resistance, and can be applied to both the multiple-permutation and single-permutation settings. (For the latter, our intuition has always been that, with a suitable matrix, lp231 should offer comparable security to LP231. But the amount of casework is much larger; analyzing lp231 by hand would require about 30 times as much paper as LP231.) The security lower bounds produced by our program are always *correct*, but are not always *tight*. For example, we suspect that a good realization of LP362 has collision security of $N^{0.6}$, but our automated analysis only proves a bound of $N^{0.55}$.

While the program is mainly designed to prove security lower bounds, it can also be used to find attacks. For example, the Joux preimage attack on $\mathrm{LP}_{231}^{\mathrm{SS}}$ (Shrimpton-Stam with permutations and feed-forward xors [28]) was located by our program before we knew of it. The program likewise discovered constant-query collision and preimage attacks on $\mathrm{lp}_{231}^{\mathrm{SS}}$. Of course it is not too hard to reconstruct such attacks with foreknowledge of their existence, but it is useful to have a program that can locate such attacks automatically. Note that because of correctness, an easily-attacked scheme is always given a poor bound by the program (at least as bad as the attack shows), but a poor bound does not necessarily indicate the program has located an attack, because the weak bound could be caused by the program's own limitations.

At this point our program can only obtain asymptotic results, not concrete numerical bounds like those in Corollaries 2 and 4. One could, theoretically, trace back through the computation and replace asymptotic bounds by numerical ones, thereby obtaining concrete numerical values. We may try to do so in the future, but it is harder than it sounds.

At least one scheme to which we applied our automated analysis, $\mathrm{LP}_{231}^{\mathrm{SS}}$, is small enough that one should be able to get numerical results by carrying out a hand analysis of the type made for LP231. For most schemes this would not be feasible, as the number of cases searched by the code is too large.

Reduction and basic ideas. The problem of finding a collision or preimage can be recast as the problem of finding a solution to the system $M\mathbf{x} = b$ where M is a matrix with constant entries in $\mathbb{F}_{2^n}$, where b is a constant vector with entries in $\mathbb{F}_{2^n}$, and where $\mathbf{x}$ is a vector to be filled with the adversary's queries and possibly containing some free variables which the adversary can arbitrarily set. For example, for LP^{A}_{231} with a matrix $A = (a_{ij})$, finding a collision is equivalent to solving the system

$$\begin{bmatrix} a_{11} & a_{12} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & a_{23} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & 0 & a_{33} & 0 & a_{34} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{11} & a_{12} & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{21} & a_{22} & 0 & a_{23} & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{31} & a_{32} & 0 & a_{33} & 0 & a_{34} & -1 & 0 \\ a_{41} & a_{42} & 0 & a_{43} & 0 & a_{44} & 0 & a_{45} & -a_{41} & -a_{42} & 0 & -a_{43} & 0 & -a_{44} & 0 & -a_{45} \end{bmatrix} \mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where $\mathbf{x} = (v_1, v_2, x_1, y_1, x_2, y_2, x_3, y_3, v'_1, v'_2, x'_1, y'_1, x'_2, y'_2, x'_3, y'_3)^{\mathsf{T}}$. (Our matrix has minus signs for readability, but these have no effect in a field of characteristic two.) Here v_1, v_2, v'_1, v'_2 are variables the adversary can choose arbitrarily, but each pair (x_i, y_i) or (x'_i, y'_i) must come from the adversary's query history. The first three rows stipulate the linear relationships between the inputs of the first word and the inputs/outputs of the permutations in that word; the next three rows do the same for the second word; and the last row stipulates a collision. Note that the adversary can obtain a trivial solution by setting $(x_i, y_i) = (x'_i, y'_i)$ for $i = 1, 2, 3$, which corresponds to a "collision" between two equal words. Since this is of course uninteresting, we require that $(x_i, y_i) \neq (x'_i, y'_i)$ for some i, or more precisely that (x_i, y_i) is a *distinct query* from (x'_i, y'_i) for some i, as two queries could have the same inputs and outputs and still be distinct (if they come from different permutations).

Since the latter condition is a bit cumbersome to work with, it is easier to assume that *all* queries in $\mathbf{x}$ are distinct, and to analyze with separate linear systems the special cases in which the queries are not all distinct. For example, for the adversary to obtain an attack with $(x_3, y_3) = (x'_3, y'_3)$ but with all other queries distinct, the adversary must solve the modified system

$$\begin{bmatrix} a_{11} & a_{12} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & a_{23} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & 0 & a_{33} & 0 & a_{34} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{11} & a_{12} & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{21} & a_{22} & 0 & a_{23} & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & a_{31} & a_{32} & 0 & a_{33} & 0 & a_{34} \\ a_{41} & a_{42} & 0 & a_{43} & 0 & a_{44} & 0 & 0 & -a_{41} & -a_{42} & 0 & -a_{43} & 0 & -a_{44} \end{bmatrix} \mathbf{x}' = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

obtained by adding the x'_3-column to the x_3-column, adding the y'_3-column to the y_3-column, and dropping the x'_3 and y'_3-columns: this accounts for setting

$x_3 = x'_3$ and $y_3 = y'_3$. Here $\mathbf{x}' = (v_1, v_2, x_1, y_1, x_2, y_2, x_3, y_3, v'_1, v'_2, x'_1, y'_1, x'_2, y'_2)^\mathsf{T}$ is $\mathbf{x}$ with x'_3 and y'_3 dropped. Thus by analyzing a separate system for each way of identifying queries, the problem reduces to analyzing the probability of solving systems of the form $M\mathbf{x} = b$ where the queries in $\mathbf{x}$ are all distinct.

Our program operates in the limit, where the entries of M and b, as well as the query coordinates themselves, come from larger and larger fields $\mathbb{F}_{2^n}$. This requires some additional explanations. Say that two matrices of M and M' of the same dimensions but over possibly different fields are *column similar* if the rank of each subset of columns of M is equal to the rank of the corresponding subset in M'. What our program effectively does is this: given a sequence of column similar matrices $(M_n)_{n \geq h}$ where each M_i is a matrix with entries in $\mathbb{F}_{2^i}$, and given an arbitrary infinite sequence $(b_n)_{n \geq h}$ and a number $\alpha \in (0, 1)$, it upper bounds the limit as $n \to \infty$ of the probability of the adversary obtaining a solution to $M_n \mathbf{x} = b_n$ if it uses no more than $q = N^\alpha$ queries. Of course we cannot input an infinite sequence $(M_n)_{n \geq h}$ or $(b_n)_{n \geq h}$; all the program ever knows about $(M_n)_{n \geq h}$ is the rank of each subset of columns of M_h, which it knows because it is given M_h itself. As for the sequence of right-hand sides $(b_n)_{n \geq h}$, its values are never used to compute the upper bound, so we do not care.

The result of the computation is obviously dependent on M_h, so we must explain how this matrix is chosen. Say that one wishes to instantiate LP^A_{mkr} with $n = 128$, so the entries of A will be points in $\mathbb{F}_{2^{128}}$. Then $M_h = M_{128}$ is a function of A, and A is simply chosen at random. One can choose the entries of A uniformly in $\mathbb{F}_{2^{128}}$, or one can limit the entries of A to numbers with few bits in order to increase the efficiency of LP^A_{mkr} (if the entries are chosen from too small a set, however, M_h can have poor linear independence properties leading to a sub-optimal bound; then A must be re-chosen).

One can significantly increase the speed of the computation by giving the program a matrix $M_{h'}$ that is column-similar to M_h but where h' is smaller than $h = 128$, such as, say, $h' = 7$, because field operations over $\mathbb{F}_{2^7}$ can be implemented more efficiently than over $\mathbb{F}_{2^{128}}$. Finding such a matrix $M_{h'}$ requires trial and error, but is generally not too hard. If $M_{h'}$ cannot be found, a second method is to generate $M_{h'}$ directly by choosing a random matrix A' with entries in $\mathbb{F}_{2^{h'}}$ and then to use a field embedding of $\mathbb{F}_{2^{h'}}$ into $\mathbb{F}_{2^{128}}$ to recover A. The only disadvantage of this method is that one does not obtain a matrix A with small entries. Sample matrices A and $M_{h'}$ used for our computations are given in the full version of this paper [23].

System constraints. The problem thus reduces to upper bounding the probability of the adversary obtaining a solution to a system $M_n\mathbf{x} = b_n$ where M_n is an unknown matrix over $\mathbb{F}_{2^n}$ column similar to a given matrix M_h with entries in $\mathbb{F}_{2^h}$ and b_n is an arbitrary vector with entries in $\mathbb{F}_{2^n}$. But, beyond this, there are also stipulations on the solution $\mathbf{x}$, certain entries of which must come from the adversary's query history. We now make this part precise.

A *system constraint* ρ on a matrix M is a partition of M's columns into a set of *free* columns, a set of *null* columns, and zero or more sets of two columns each called *query pairs*. Each query pair of has a *first* and *second* column. When we multiply M by a column vector $\mathbf{x}$ its entries inherit ρ's partition of the columns of M, so we can speak of the "free" or "null" entries of $\mathbf{x}$, and of pairs of entries of $\mathbf{x}$ that form query pairs. If M_n is a matrix with entries in $\mathbb{F}_{2^n}$ and b_n is a vector with entries in $\mathbb{F}_{2^n}$ and $\mathcal{Q} = \mathcal{Q}(A)$ is the query history of an adversary interacting with ideal-permutation oracles of domain (and range) $\mathbb{F}_{2^n}$, then we say that $\mathcal{Q}$ *solves* $M_n\mathbf{x} = b_n$ *with respect to* ρ, or that $\mathcal{Q}$ *solves* $(M_n\mathbf{x} = b_n, \rho)$, if there exists a vector $\mathbf{x}$ with entries in $\mathbb{F}_{2^n}$ such that: (i) every null entry of $\mathbf{x}$ is 0, (ii) if (x, y) is a query pair of $\mathbf{x}$ with first element x and second element y then (i, x, y) is a query in $\mathcal{Q}$ for some π_i, and (iii) no two query pairs of $\mathbf{x}$ correspond to the *same* query in $\mathcal{Q}$. If ρ is understood we simply say that $\mathcal{Q}$ *solves* $M_n\mathbf{x} = b_n$.

Note that in the above definition we do not attach any importance to which query pairs are mapped to which kind of queries in $\mathcal{Q}$; in fact, it does not even matter for the definition how many oracles the adversary is interacting with. All that matters is that when the adversary makes a query to one of its oracles, the resulting output comes uniformly at random from a pool of size at least $N' = N - q$. As always, the adversary can make both forward and backward queries.

From here on it will simplify the discussion if we assume there is a single matrix M and vector b under discussion, which implicitly depend on n. However one should remember that all one knows about M is the rank of any subset of columns of M, and that b is arbitrary.

INDUCTION. Note that it is trivial to tell whether the adversary can solve $(M\mathbf{x} = b, \rho)$ if ρ has no query pairs, since in this case the problem reduces to the feasibility of a standard system of linear equations. Determining the probability of the adversary solving $(M\mathbf{x} = b, \rho)$ becomes increasingly complicated as ρ has more query pairs. To upper bound the probability of the adversary solving $(M\mathbf{x} = b, \rho)$ the program works by induction on the number of query pairs in ρ, using bounds on the probability of solving systems $(M\mathbf{x} = b, \rho')$ where ρ' has fewer query pairs than ρ.

What is ultimately of interest is whether for a given $\alpha \in (0, 1)$ the adversary has zero probability, in the limit, of finding *some* solution to the system in $q = N^\alpha$ queries (consider α as fixed throughout). But to get the induction to work we also need to know, if the adversary has nonzero chance of solving the system in the limit, *how many* solutions the adversary can reasonably expect to get. Here the "number" of solutions of a system $(M\mathbf{x} = b, \rho)$ with respect to a query history $\mathcal{Q}$ is counted in a special way: solutions $\mathbf{x}$ and $\mathbf{x}'$ only count as distinct if the values of some query pair are different in them (meaning they are mapped to distinct queries of $\mathcal{Q}$). If ρ has no query pairs at all, then the number of solutions is defined as 1.

The following definition formalizes the idea of "how many solutions the adversary can reasonably expect to get". It is central to the analysis.

Definition 1. *Let $\alpha \in (0,1)$. A number $\beta \geq 0$ is an α-**threshold** of $(M\mathbf{x} = b, \rho)$ if for any $\epsilon > 0$, the probability that there exists some c for which the adversary obtains at least $N^{\beta+\epsilon}$ solutions to $(M\mathbf{x} = c, \rho)$ in $q = N^{\alpha}$ queries goes to zero as $n \to \infty$.*

Observe that b is immaterial in the definition. Also note the α-threshold is not unique since every number greater than an α-threshold is also an α-threshold. On the other hand, it is easy to see that the set of α-thresholds is a half-closed interval: if $\beta + \epsilon$ is an α-threshold for every $\epsilon > 0$, then β is also an α-threshold.

As an example, consider the system $M\mathbf{x} = b$ where M consists of a single row and $\mathbf{x} = (x, y)^{\mathsf{T}}$ has two variables, so that the system can be written

$$m_{11}x + m_{12}y = b_1 \tag{3}$$

where b_1 is the unique entry of b and where $[m_{11}\ m_{12}] = M$. Take ρ to be the system constraint on M with a unique query pair consisting of (m_{11}, m_{12}) (so ρ has no free columns or null columns), so that solving $(M\mathbf{x} = b, \rho)$ means getting a query (i, x, y) such that (x, y) solves (3). If $m_{11}, m_{12} \neq 0$ one can show that the probability of finding such a query in $q = N^{\alpha}$ queries where $\alpha \in (0,1)$ goes to 0 as $n \to \infty$ (each query made has chance $1/N' \approx 1/N$ of solving the system, so a sum bound shows the chance of success is at most N^{α}/N, which is $\ll 1$ for large n). Furthermore, one can show the probability of there being a $c \in \mathbb{F}_{2^n}$ such that $(M\mathbf{x} = c, \rho)$ has many solutions—say more than N^{ϵ}—approaches zero as $n \to \infty$, for fixed $\epsilon > 0$ (otherwise put, the set of values $\{m_{11}x + m_{12}y : (i, x, y) \in \mathcal{Q}(A)\}$ is "well spread-out" in $\mathbb{F}_{2^n}$). By definition, this means that $(M\mathbf{x} = b, \rho)$ has zero α-threshold.

In the above example the α-threshold of $(M\mathbf{x} = b, \rho)$ does not depend on α. As a second example, consider a system $M\mathbf{x} = b$ where M is a 1×4 matrix, written

$$m_{11}x_1 + m_{12}y_1 + m_{13}x_2 + m_{14}y_2 = b_1. \tag{4}$$

We assume each m_{1j} is nonzero. Let ρ partition the columns of M into two query pairs, (m_{11}, m_{12}) and (m_{13}, m_{14}). Then one can show that $\max(0, 2\alpha - 1)$ is an α-threshold of $(M\mathbf{x} = b, \rho)$. (Intuitively, each query made, whether forward or backward, gives a random value of $m_{11}x_1 + m_{12}y_1$ and a random value of $m_{13}x_2 + m_{14}y_2$; thus q queries give q^2 pairs of random values; for any $c \in \mathbb{F}_{2^n}$ the expected number of such pairs that sum to c is $q^2/N = N^{2\alpha-1}$; one can finally show that the probability of there being a c with more than $N^{\epsilon}N^{\max(0,2\alpha-1)}$ pairs summing to it approaches zero as $n \to \infty$ for fixed $\epsilon > 0$, which by definition means that $\max(0, 2\alpha - 1)$ is an α-threshold of $(M\mathbf{x} = b, \rho)$.) Thus, for example, if an adversary makes $q = N^{\alpha} = N^{3/4}$ queries there is negligible chance there will be some c for which $(M\mathbf{x} = c, \rho)$ has more than $N^{1/2+\epsilon}$ solutions, for fixed $\epsilon > 0$ and as $n \to \infty$.

The crux of the problem lies in knowing how to compute an α-threshold for a system $(M\mathbf{x} = b, \rho)$ given α-thresholds for every system $(M\mathbf{x} = b, \rho')$ where ρ' has fewer query pairs than ρ. There are a few ways for doing this, possibly giving different numbers (the method giving the smallest result is the better, obviously). We will sketch some of the methods. Note that if ρ has no query pairs then $(M\mathbf{x} = b, \rho)$ has a zero α-threshold, by our earlier convention on how to count the number of solutions to a system.

The induction step. Take a system $(M\mathbf{x} = b, \rho)$. Say each system $(M\mathbf{x} = b, \rho')$ where ρ' has fewer query pairs than ρ has a (previously computed) α-threshold $\beta_{\rho'}$. We can assume the adversary has found at most $N^{\beta_{\rho'}+\epsilon}$ solutions for each system $(M\mathbf{x} = b, \rho')$. Here $\epsilon > 0$ can be as small as desired. Making these assumptions costs us a small additive constant for each ρ', because it *is* possible for the adversary to obtain more than $N^{\beta_{\rho'}+\epsilon}$ solutions to $(M\mathbf{x} = c, \rho')$ for some c and ρ', if it is lucky, but, by the definition of an α-threshold, the sum of these additive constants goes to zero as $n \to \infty$.

We introduce some more notation. We write $(M\mathbf{x} = b, \rho, \mathcal{Q})$ to emphasize that the set of solutions of $(M\mathbf{x} = b, \rho)$ depends on $\mathcal{Q}$. Also recall that two solutions of $(M\mathbf{x} = b, \rho, \mathcal{Q})$ are counted as distinct only if the two solutions map some query pair of ρ to different queries in $\mathcal{Q}$; a good way to think of the set of solutions is that each solution only specifies the values for the query pairs of ρ, the free variables being left unspecified. We write the elements of $\mathcal{Q}$ as tuples (x, y) rather than as triplets (i, x, y) since the index of the permutation queried does not matter in the model.

One can first observe that, for any solution $\mathbf{x}$ of $(M\mathbf{x} = b, \rho, \mathcal{Q})$, one of the queries in $\mathcal{Q}$ used by $\mathbf{x}$ comes last in chronological order in $\mathcal{Q}$. We say this query *creates* the solution. A natural way to bound the probability of the adversary obtaining many solutions to $(M\mathbf{x} = b, \rho)$ is to compute the probability that any given query creates a solution, and to consider that only q queries are made in total, so that only so many solutions can be expected. To further break down the problem one can evaluate separately, for each query pair in ρ, the probability that a given query creates a solution where the query is matched *to that pair*.

So take a query pair (C_1, C_2) in ρ, where C_1, C_2 are the two columns of the query pair. Say the adversary has already made a certain sequence $\mathcal{Q}'$ of queries, and is now making, say, a new forward query $\pi(x)$ to one of its oracles; we want to know the probability that the resulting output $y = \pi(x)$ creates a solution $\mathbf{x}$ of $(M\mathbf{x} = b, \rho, \mathcal{Q})$ where $\mathcal{Q} = \mathcal{Q}' \cup \{(x, y)\}$ and where the coefficients x, y are used for the columns C_1, C_2 respectively (to denote this, we will say that the query (x, y) is *used in position* (C_1, C_2)). We distinguish between the case when C_2 is linearly independent of the free columns of ρ and the case when it is not.

Say first that C_2 is linearly independent of the free columns of ρ. Let ρ' be the system constraint obtained from ρ by changing C_2 into a free column and changing C_1 into a null column. Also let $b' = b - xC_1$. Then for each solution of $(M\mathbf{x} = b, \rho, \mathcal{Q})$ where (x, y) is used in position (C_1, C_2) there is exactly one solution of $(M\mathbf{x} = b', \rho', \mathcal{Q}')$. Moreover, for every solution $\mathbf{x}$ in $(M\mathbf{x} = b', \rho', \mathcal{Q}')$ there is *at most one* value of y for which the query (x, y) extends the solution $\mathbf{x}$ to a solution of $(M\mathbf{x} = b, \rho, \mathcal{Q})$ where (x, y) is used in position (C_1, C_2). This is because C_2 is linearly independent from the free columns of ρ, so that when the coefficients of all columns of M have been fixed except for the coefficient of C_2

and the coefficients of the free columns of ρ, there is at most one coefficient for C_2 for which the remaining linear system will have a solution for a given right-hand side. So the number of successful outputs y is at most the number of solutions of $(M\mathbf{x} = b', \rho', \mathcal{Q}')$.

If ρ' has an α-threshold β, then the probability the returned value y will give a solution of $(M\mathbf{x} = b, \rho, \mathcal{Q})$ where (x, y) is used in position (C_1, C_2) is thus at most $N^{\beta+\epsilon}/N'$, since $(M\mathbf{x} = b', \rho', \mathcal{Q}')$ has at most $N^{\beta+\epsilon}$ solutions and since y is returned uniformly at random from a set of size at least N'. Since the adversary makes $q = N^{\alpha}$ queries in total the "expected" number of solutions is $N^{\alpha+\beta+\epsilon}/N'$. There are two cases to distinguish: $\alpha + \beta \leq 1$ and $\alpha + \beta > 1$. For the first case, the expected number of solutions is $\leq N^{\epsilon}$ and it isn't hard to show that the probability that there exists a c for which the adversary obtains more than $N^{2\epsilon}$ solutions to $(M\mathbf{x} = c, \rho)$ in this way goes to zero as $n \to \infty$. If $\alpha + \beta > 1$, a similar argument shows that the probability of the adversary obtaining more than $N^{\alpha+\beta+2\epsilon-1}$ solutions to $(M\mathbf{x} = c, \rho)$ for some c in this way goes to 0 as $n \to \infty$. If the adversary's only means of constructing solutions to $(M\mathbf{x} = b, \rho)$ was to fill in last the values of the query pair (C_1, C_2) with a forward query, one would thus obtain an α-threshold of $\max(0, \alpha + \beta - 1)$ for $(M\mathbf{x} = b, \rho)$, because $\epsilon > 0$ is arbitrary. But of course there are the other possibilities to consider: backward queries and solutions were the last query pair filled in is not (C_1, C_2). (In the end, the maximum α-threshold is retained unless some query pair column of ρ is linearly dependent on the free columns of ρ; see the next case.)

In the second case the column C_2 is linearly dependent on the free columns of ρ. In this case let ρ' be obtained from ρ by setting to "null" both C_1 and C_2, and let $b' = b - xC_1$. Then the number of solutions of $(M\mathbf{x} = b, \rho, \mathcal{Q})$ where (x, y) is in position (C_1, C_2) is equal to the number of solutions of $(M\mathbf{x} = b', \rho', \mathcal{Q}')$. Let β be the α-threshold of $(M\mathbf{x} = b, \rho')$. Since there are at most $N^{\beta+\epsilon}$ solutions to $(M\mathbf{x} = b', \rho')$, and since there are at most $q = N^{\alpha}$ queries (x, y) in the query history, the total number of solutions of $(M\mathbf{x} = b, \rho)$ obtained in q queries is at most $N^{\alpha+\beta+\epsilon}$. Because $\epsilon > 0$ is arbitrary, $\alpha + \beta$ is therefore an α-threshold of $(M\mathbf{x} = b, \rho)$. Note that unlike in the first case, we do not need to examine any other query pairs of ρ to establish this threshold. This benefits the program's speed, as having the computation of ρ's α-threshold depend only on one other α-threshold instead of depending on several other α-thresholds helps stave off a combinatorial explosion of cases.

A few other techniques for computing α-thresholds recursively, which sometimes give better bounds, are used by the program. These are discussed in the paper's full version [23]

Final probability of success. What ultimately interests us is the adversary's probability of obtaining *some* solution to the system $(M\mathbf{x} = b, \rho)$ where ρ is the original "root" system constraint with the maximum number of query pairs and b is the original right-hand side. If, after applying the recursion, the system $(M\mathbf{x} = b, \rho)$ is found to have a nonzero α-threshold, then we cannot conclude anything; the program has effectively failed to show the scheme requires at least

$q = N^\alpha$ queries to break. But if $(M\mathbf{x} = b, \rho)$ has a zero α-threshold, then for any $\epsilon > 0$ the probability of obtaining at least one solution of $(M\mathbf{x} = b, \rho)$ in $q = N^{\alpha-\epsilon}$ queries goes to 0 as $n \to \infty$. Seeing so requires revisiting how the α-thresholds are obtained. From the recursion process, there is only one way that a system constraint can have zero α-threshold: when, at every query, the adversary has probability at most $N^{\beta+\epsilon'}/N'$ of obtaining a solution for some β with $\alpha+\beta \leq 1$ (see the first case considered for the computation of α-thresholds). By choosing $\epsilon' < \epsilon$, then, we see that the adversary's final probability of success is bounded by the probability of solving the original system in $q = N^{\alpha-\epsilon}$ queries where each query has chance at most $N^{\beta+\epsilon'}/N'$ of giving a solution; but since $\alpha-\epsilon+\beta+\epsilon' < 1$, a sum bound directly shows that this probability of success goes to 0 as $n \to \infty$. Thus the scheme asymptotically requires at least N^α queries to be broken. To find the best α the program simply does a binary search.

Results. Findings produced using our program are presented in Fig. 3. Some of the cases are solved quickly, but others take over an hour and involve the exploration of millions of system constraints.

5 Discussion

It may be interesting to compare the efficiency of LP compression functions and conventional blockcipher-based ones. It is conventional to use *rate* as a rough measure of efficiency, but the rate of a blockcipher-based construction, as conventionally defined [17, p. 340], doesn't even attend to the number of key bits employed. The simplest way to correct this is to say that the *adjusted rate* of a blockcipher-based hash-function is the number of message bits processed per blockcipher input bits, the latter including plaintext bits and key bits (for simplicity, equally weighted). Then SHA-1 would have an adjusted rate of 0.76; Davies-Meyer [16], 0.5; MDC-2 [16], 0.27; Hirose's double-length construction [11], 0.17; and MDC-4, 0.13. From this vantage, the adjusted rate of LP231, 0.33, and LP362, 0.17, are competitive. Regardless, adjusted rate is a coarse measure of efficiency, and the current work aims only to probe the security and feasibility of LP compression functions, not to vanquish any other design.

This paper has only dealt with making a compression function, not a full-fledged hash function. Of course you can always turn the former into the latter using Merkle-Damgård [9, 18] or any of the other techniques that have emerged in recent years [1, 3, 10, 22], but the "best" approach remains to be seen. Also, we have considered only collision and preimage resistance. Certainly there are other desirable properties one should aim for in a contemporary construction, like being indifferentiable from a random oracle [8].

Acknowledgments

Most of the work on this paper was carried out while the second author was in the Department of Mathematics at UC Davis. Both authors received funding from NSF grant CCR-0208842 and a gift from Intel; many thanks to Intel, particularly Jesse Walker, and to the NSF, for their kind support.

References

1. E. Andreeva, G. Neven, B. Preneel, and T. Shrimpton. Seven-property preserving iterated hashing: ROX. *ASIACRYPT 2007*. LNCS vol. 4833, Springer, pp. 130–146, 2007.
2. M. Bellare and T. Ristenpart. Hash functions in the dedicated-key setting: design choices and MPP transforms. *International Colloquium on Automata, Languages, and Programming – ICALP 2007*. LNCS vol. 4596, Springer, pp. 399–410, 2007.
3. M. Bellare and T. Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. *ASIACRYPT 2006*. LNCS vol. 4284, Springer, pp. 299–314, 2006.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. *EUROCRYPT 2008*. LNCS vol. 4965, Springer, pp. 181–197, 2008.
5. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. *Ecrypt Hash Workshop 2007*. Available at http://sponge.noekeon.org/
6. J. Black, M. Cochran, and T. Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. *EUROCRYPT 2005*. LNCS vol. 3494, Springer, pp. 526–541, 2005.
7. J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash function constructions from PGV. *CRYPTO 2002*. LNCS vol. 2442, Springer, pp. 320–335, 2002.
8. J. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: how to construct a hash function. *CRYPTO 2005*. LNCS vol. 3621, pp. 430–448, 2005.
9. I. Damgård. A design principle for hash functions. *CRYPTO '89*. LNCS vol. 435, Springer, pp. 416–427, 1990.
10. Y. Dodis, K. Pietrzak, and P. Puniya. A new mode of operation for block ciphers and length-preserving MACs. *EUROCRYPT 2008*. LNCS vol. 4965, Springer, pp. 198–219, 2008.
11. S. Hirose. Some plausible construction of double-block-length hash functions. *Fast Software Encryption*. LNCS vol. 4047, pp. 210–225, Springer, 2006.
12. M. Hattori, S. Hirose, and S. Yoshida. Analysis of double block length hash functions. *Cryptography and Coding, 9th IMA International Conference Coding*. LNCS vol. 2898, Springer, pp. 290–302, 2003.
13. A. Joux. Multicollisions in iterated hash functions. *CRYPTO 2004*. LNCS vol. 3152, Springer, pp. 306–316, 2004.
14. L. Knudsen, X. Lai, and B. Preneel. Attacks on fast double block length hash functions. *Journal of Cryptology*, 11(1), pp. 59–72, 1998.
15. S. Lucks. A failure-friendly design principle for hash functions. *ASIACRYPT 2005*. LNCS vol. 3788, Springer, pp. 474–494, 2005.
16. S. Matyas, C. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Tech. Disclosure Bulletin*, 27, pp. 5658–5659, 1985.
17. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
18. R. Merkle. One way hash functions and DES. *CRYPTO '89*. LNCS vol. 435, Springer, pp. 428–446, 1990.
19. M. Nandi. Designs of efficient secure large hash values. Cryptology ePrint report 2005/296.
20. B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: a synthetic approach. *CRYPTO '93*. LNCS vol. 773, Springer, pp. 368–378, 1994.

21. B. Preneel, R. Govaerts, and J. Vandewalle. On the power of memory in the design of collision resistant hash functions. *ASIACRYPT '92*. LNCS vol. 718, Springer, pp. 105–121, 1993.
22. T. Ristenpart and T. Shrimpton. How to build a hash function from any collision-resistant function. *ASIACRYPT 2007*. LNCS vol. 4833, Springer, pp. 147–163, 2007.
23. P. Rogaway and J. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. Full version of this paper. Manuscript, 2008. Available from either author's web page.
24. P. Rogaway and J. Steinberger. Security/efficiency tradeoffs for permutation-based hashing. *EUROCRYPT 2008*. LNCS vol. 4965, Springer, pp. 220–236, 2008.
25. T. Peyrin, H. Gilbert, F. Matthew, and J. Robshaw. Combining compression functions and block cipher-based hash functions. *ASIACRYPT 2006*. LNCS vol. 4284, Springer, pp. 315–331, 2006.
26. T. Satoh, M. Haga, and K. Kurosawa. Towards secure and fast hash functions. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, pp. 55–62, 1999.
27. C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, vol. 27, pp. 379–423 and pp. 623–656, 1948.
28. T. Shrimpton and M. Stam. Building a collision-resistant compression function from non-compressing primitives. *International Colloquium on Automata, Languages, and Programming – ICALP 2008*. LNCS, Springer, 2008. Earlier version presented at the *ECRYPT Hash Workshop*, May 2007.
29. M. Stam. Beyond uniformity: better security/efficiency tradeoffs for compression function security. *CRYPTO 2008*. LNCS, Springer, 2008.
30. J. Steinberger. The collision intractability of MDC-2 in the ideal-cipher model. *EUROCRYPT 2007*. LNCS vol. 4515, Springer, pp. 34–51, 2007.