A New Class of Invertible Mappings

Alexander Klimov and Adi Shamir

Computer Science department, The Weizmann Institute of Science
Rehovot 76100, Israel
{ask,shamir}@wisdom.weizmann.ac.il

Abstract. Invertible transformations over n-bit words are essential ingredients in many cryptographic constructions. When n is small (e.g., $n = 8$) we can compactly represent any such transformation as a lookup table, but when n is large (e.g., $n = 64$) we usually have to represent it as a composition of simpler operations such as linear mappings, S-P networks, Feistel structures, etc. Since these cryptographic constructions are often implemented in software on standard microprocessors, we are particularly interested in invertible univariate or multivariate transformations which can be implemented as small compositions of basic machine instructions on 32 or 64 bit words. In this paper we introduce a new class of provably invertible mappings which can mix arithmetic operations (negation, addition, subtraction, multiplication) and boolean operations (not, xor, and, or), are highly efficient, and have desirable cryptographic properties. In particular, we show that for any n the mapping $x \to x + (x^2 \vee C) \ (mod\ 2^n)$ is a permutation with a single cycle of length 2^n iff both the least significant bit and the third least significant bit in the constant C are 1.

1 Introduction

Block ciphers are among the most useful constructions in modern cryptography. They are usually constructed by repeatedly applying a simple invertible round function, which should be nonlinear with good avalanche properties. Since most block ciphers are now implemented in software on standard microprocessors, it is crucial to choose a round function which can be efficiently implemented with very few machine instructions. The importance of software-based efficiency was clearly demonstrated during the AES selection process [9] [11].

Since modern microprocessors have extremely fast arithmetic and boolean operations on 32 or 64 bit words, we would like to choose round functions which use such operations on complete words rather than operations that manipulate individual bits or bytes. In addition, it is a good idea to mix arithmetic and boolean operations in order to avoid pure algebraic or bit-oriented attacks, and to reduce the probabilities of linear and differential attacks.

The invertibility of round functions is of course a necessary condition in block ciphers, but it is also important in other constructions such as stream ciphers and hash functions: If we repeatedly apply a round function to the internal state, we want to prevent an incremental loss of entropy which can lead to early looping

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 470–483, 2003.

of the output stream or to undesirable collisions among the hashed values. In these applications we do not actually want to run the round function backwards — we just need a guarantee that the forward mapping is one-to-one. In other applications such as the encryption of data stored on backup tapes, we have to encrypt huge amounts of data once a day, but rarely decrypt the result. In these cases the software efficiency of the backward mapping is relatively unimportant, and we can use slightly asymmetric schemes in which the process of decryption is slower (by a small factor) than the process of encryption. Such an asymmetry can even help against some cryptanalytic attacks if the opponent only knows the ciphertext and tries to perform exhaustive decryption under all possible keys.

The goal of this paper is to design new types of provably invertible round functions with good cryptographic properties and extremely efficient software implementations. They mix a small number of arithmetic and boolean operations on full machine words, and cannot be inverted by executing the same program backwards with inverse operations. Instead, we can use the mathematical proof of invertibility in order to construct a completely different (and somewhat slower) inversion algorithm, if we ever have to run the mapping backwards.

The best way to introduce the new idea is to describe several simple examples which demonstrate the fine line between invertible and noninvertible mappings. Let 1 denote the constant word $0 \cdots 01$, and assume that all the operations are carried out on n-bit words. Then the following univariate mappings are invertible:

$$x \to x + 2x^2, \quad x \to x + (x^2 \vee 1), \quad x \to x \oplus (x^2 \vee 1)$$

whereas the following closely related variants are noninvertible:

$$x \to x + x^2, \quad x \to x + (x^2 \wedge 1), \quad x \to x + (x^3 \vee 1)$$

The same techniques can be used to prove the invertibility of multivariate mappings such as:

$$(x, y) \to (x \oplus 2(x \wedge y), (y + 3x^3) \oplus x).$$

2 Notations and Definitions

In this section we introduce our notations and basic definitions. We denote the set $\{0, 1\}$ by $\mathbb{B}$. We use the same symbol x to denote the n-bit vector $([x]_{n-1}, [x]_{n-2}, \cdots, [x]_0)$ in $\mathbb{B}^n$ and an integer modulo 2^n, with the usual conversion rule: $x \longleftrightarrow \sum_{i=0}^{n-1} 2^i [x]_i$.

The basic operations we allow in our constructions are the following arithmetic and boolean operations:

Definition 1. *Let x and y be n-bit input variables. A function $\phi : \mathbb{B}^{k \times n} \to \mathbb{B}^n$ is called a* **primitive function** *if $k = 1$ and $\phi(x)$ is one of the operations of negation: $\phi(x) = -x \pmod{2^n}$ and complementation: $[\phi(x)]_i = \overline{[x]}_i$, or if $k = 2$ and $\phi(x, y)$ is one of the operations of addition: $\phi(x, y) = x + y \pmod{2^n}$, subtraction: $\phi(x, y) = x - y \pmod{2^n}$, multiplication: $\phi(x, y) = x \cdot y \pmod{2^n}$, xor: $[\phi(x, y)]_i = [x]_i \oplus [y]_i$, and: $[\phi(x, y)]_i = [x]_i \wedge [y]_i$, or: $[\phi(x, y)]_i = [x]_i \vee [y]_i$.*

Note that left shift is allowed (since it is equivalent to multiplication by a power of two), but right shift and circular rotations are not allowed in our framework, even though they are available as basic machine instructions in most microprocessors.

When we deal with an $m \times n$ bit matrix p, we start numbering its rows and columns from zero, and refer to its i-th row as $p_{i-1,\star}$ and to its j-th column as $p_{\star,j-1}$. Our multivariate transformations operate on inputs represented by the bit matrix $p_{\star,\star}$ (for plaintexts) and produce outputs represented by the matrix $c_{\star,\star}$ (for ciphertexts), where the value of the i-th variable is represented by the successive bits in row $i-1$ in the matrix.

Definition 2. Parametric functions *are functions* $g(x_1, \cdots, x_a; \alpha_1, \cdots, \alpha_b)$ *whose arguments are split by a semicolon into* **inputs** *(the* x_i*'s) and* **parameters** *(the* α_j*'s). A* **Parametric Invertible Function (PIF)** *is a parametric function whose input/output relationship is invertible for any fixed value of the parameters:*

$$\forall \alpha \; \forall x, y : g(x;\alpha) = g(y;\alpha) \iff x = y$$

3 Previous Constructions of Invertible Mappings

To test the applicability of previous construction techniques, we will try to use each technique in order to prove the claim that the univariate mapping $x \rightarrow x \oplus (x^2 \vee 1)$ over n-bit words is invertible for any n.

The simplest way to construct a large class of invertible mappings is to compose other invertible operations in various orders. A typical example is an S-P network in which we alternately apply invertible substitution and permutation operations. However, in our running example the global invertibility of the mapping $x \rightarrow x \oplus (x^2 \vee 1)$ cannot be explained by the local invertibility of its basic operations, since squaring and or'ing with 1 are non-invertible operations.

A second technique for constructing invertible transformations was proposed by Feistel [3] in 1973 and used in many block ciphers such as DES [7]. Unlike the first technique, it can use arbitrary non-invertible functions f in the construction by using the PIF $g(l;r) = l \oplus f(r)$. The basic Feistel construction maps two inputs (l, r) into $(r, l \oplus f(r))$, and the full Feistel construction iterates this mapping any number of times. This idea was generalized in several directions, and in [13] Schneier and Kelsey gave "a taxonomy of Feistel networks" which include *unbalanced Feistel networks* (UFN), *homogeneous* and *heterogeneous* UFNs, *incomplete* and *inconsistent* networks, etc. However, the Feistel construction requires at least two variables, and thus it also fails to explain the invertibility of our running example $x \rightarrow x \oplus (x^2 \vee 1)$ $(mod\ 2^n)$.

A third method of constructing invertible multivariate transformations is called triangulation, and is motivated by the way we use Gauss elimination to solve a system of linear equations by first triangulating its matrix of coefficients, and then computing the values of the variables in a sequential way. For example, in 1993 Shamir proposed [14] two constructions of birational transformations, in

which both the mapping and its inverse are defined by ratios of small polynomials. The core of his first scheme was the fact that the mapping

$$(x_0, \ldots, x_{k-1}) \mapsto (f_0(x_0, \ldots, x_{k-1}), \ldots, f_{k-1}(x_0, \ldots, x_{k-1})) \quad (1)$$

is uniquely invertible for almost all inputs if each f_i has the following form:

$$f_i(x_0, \ldots, x_{k-1}) = g_i(x_0, \ldots, x_{i-1})x_i + h_i(x_0, \ldots, x_{i-1}) \pmod{N}, \quad (2)$$

where g_i and h_i are arbitrary non-zero polynomial functions modulo a large RSA modulus N. The inversion process solves a series of trivial linear equations in one variable, as demonstrated in the following mapping from (x, y, z) to (x', y', z'):

$$\begin{aligned} x' &= 7x \pmod{N} \\ y' &= (x^2 + 12x + 7)y + x \pmod{N} \\ z' &= (xy + 4x^3 + y)z + xy^2 + y^3 \pmod{N} \end{aligned} \quad (3)$$

Given the output (x', y', z'), we can derive $x = \frac{x'}{7} \pmod{N}$ from the first equation, then use it to derive $y = \frac{y'-x}{x^2+12x+7}$ from the second equation, and finally use x and y to derive $z = \frac{z'-(xy^2+y^3)}{xy+4x^3+y}$ from the third equation. The derivation fails only when one of the numeric denominators is not relatively prime to N, which is an extremely rare event. This approach could easily handle arbitrary functions g_i and h_i which mix arithmetic and boolean operations, since these functions have to be evaluated only in the forward direction. However, it cannot explain the invertibility of our univariate running example, or the invertibility of the bivariate example $(x, y) \rightarrow (x \oplus 2(x \wedge y), (y + 3x^3) \oplus x)(mod\ 2^n)$, which can not be simplified into a triangular form.

A fourth approach is to concentrate on mappings which contain only the arithmetic operations of addition, subtraction and multiplication. Such polynomial mappings have a rich algebraic structure, and their invertible cases are called permutation polynomials. The problem of characterizing permutation polynomials over various domains is a very well studied problem in mathematics. For example, Hermite made considerable progress in characterizing univariate permutation polynomials modulo a prime p, and Dickson found an effective characterization for all univariate polynomials with degrees smaller than 5. However, even today the problem is not completely resolved for high degree polynomials modulo a large prime p.

Over the ring of integers modulo 2^n the problem seems to be much simpler. In 1997, the designers of the AES candidate RC6 [11] were looking for an invertible mapping with good mixing properties which could be implemented in software with a small number of arithmetic operations on 32 bit words. They chose the mapping $x \rightarrow x + 2x^2\ (mod\ 2^{32})$, and used an ad-hoc algebraic technique to prove that it is a permutation polynomial. In 1999, Rivest [12] generalized this proof technique and provided the following complete characterization of all the univariate permutation polynomials modulo 2^n:

Theorem 1. *Let $P(x) = a_0 + a_1x + \cdots + a_dx^d$ be a polynomial with integral coefficients. Then $P(x)$ is a permutation polynomial modulo 2^n, $n > 2$ if and only if a_1 is odd, $(a_2 + a_4 + \cdots)$ is even, and $(a_3 + a_5 + \cdots)$ is even.*

Unfortunately, his algebraic proof technique cannot be generalized to mappings which mix arithmetic and boolean operations, and thus it can not prove the invertibility of $x \to x \oplus (x^2 \vee 1)$ or of $(x, y) \to (x \oplus 2(x \wedge y), (y + 3x^3) \oplus x)$.

4 The New Construction

To understand the new construction, assume that each input variable has n bits, and we place the m input variables in the m rows of an $m \times n$ bit matrix. To compute the mapping in the forward direction, we apply to the rows of this matrix a sequence of primitive machine instructions. To compute the mapping in the backward direction, we cannot sequentially undo the effects of the row operations on the bit matrix, since some of them can be non-invertible. Instead, we derive the backward mapping by analysing the overall effect of the forward mapping on the columns of the bit matrix. Note that our goal is not to compute the inverse mapping, but only to prove that it is uniquely defined for any output matrix. Consequently, we don't actually have to perform the time consuming operation of changing the way bits are packed into words from row major to column major representation.

The idea of switching from row operations to column operations was used in 1997 by Eli Biham [2] to speed up the computation of DES. He placed 64 plaintexts in a 64×64 bit matrix, and encrypted all of them simultaneously on an alpha microprocessor (with 64 bit words) by operating on the vertical bit slices instead of on the horizontal plaintexts in the matrix. The row-oriented and column-oriented algorithms look completely different (in particular, one of them uses table lookups while the other uses only boolean operations to implement the S-boxes), but they have the same overall effect. In Biham's paper both the encryption and the decryption operations are carried out on the columns of the matrix, whereas in our case the forward mapping is performed on the rows and the backward mapping is performed on the columns of the matrix. The crucial observation which makes our construction possible is the fact that all the primitive functions can be applied to truncated versions of their inputs and provide truncated versions of the original outputs. To formalize this notion, we need the following definition:

Definition 3. *A function f from $\mathbb{B}^{m \times n}$ to $\mathbb{B}^{l \times n}$ is called a* **T-function** *if the k-th column of the output $[\phi(x)]_{\star,k-1}$ depends only on the first k columns of the input: $[x]_{\star,0}, \ldots, [x]_{\star,k-1}$.*

Consider, for example, the addition function: $x + y = z \pmod{2^n}$. The least significant bit of the result depends only on the least significant bits of the operands: $[z]_0 = [x]_0 \oplus [y]_0$. The second bit depends on the first and second bits of the operands $[z]_1 = [x]_1 \oplus [y]_1 \oplus \alpha$, where α is the carry into the second bit position which is defined by the least significant bits of x and y. The same holds

for all the other bits — in order to calculate bit number k of the result we only need to know bits $0, 1, \ldots, k$ of the operands. This is also true for subtraction and multiplication, and trivially true for all the bit-oriented boolean operations. We can thus conclude that all the primitive functions we allow in our constructions are T-functions, whereas the excluded operations of bit rotations or right shifts are not T-functions. The composition of two T-functions is also a T-function, and thus any mapping defined by a sequence of primitive functions applied to the input bit matrix is also a T-function.

The name T-function refers to the triangular dependence between the columns of the operands. Note that this **implicit triangulation** is different from the **explicit triangulation** in multivariate mappings which was mentioned in the previous section. In fact, explicit and implicit triangulations are row/column dual properties: A mapping has an explicit triangular shape if its i-th variable (row) is combined only with previous variables in the given expressions, and has an implicit triangular shape if its i-th bit slice (column) is combined only with previous bit slices by the T-functions.

To demonstrate the new construction, let us prove the following result:

Theorem 2. *For any composition f of primitive functions, the mapping $x \to x + 2 \cdot f(x)$ $(mod\ 2^n)$ is invertible.*

Note that this class includes the RC6 function $x \to x + 2x^2$ $(mod\ 2^n)$), along with more complicated examples such as $x \to x + 2 \cdot ((x \wedge x^2) \vee (\bar{x} \wedge x^3))$ in which the bits of x are used to select between the bits of x^2 and x^3.

Proof. Our goal is to recover x from a given value of $y = x + 2 \cdot f(x)$ $(mod\ 2^n)$. Assume that we already know bits $0, 1, \ldots, i-1$ of x, and our goal is to compute bit number i. Since f is a composition of primitive functions, we can calulate bits $0, 1, \ldots, i-1$ of $f(x)$ by evaluating all the arithmetic operations in f modulo 2^i, and truncating all the boolean operations after bit $i-1$. By leftshifting this result, we obtain all the values of bits $0, 1, \ldots, i$ of $2 \cdot f(x)$. When we reduce the equation $y = x + 2 \cdot f(x)$ modulo 2^{i+1}, the only unknown bit in the truncated equation is the i-th bit in the first occurence of x in this expression, and thus we can easily derive it from the equation. By repeating this process, we can recover all the bits of x from lsb to msb in a unique way for any f which is a composition of primitive functions.

Mappings defined by T-functions make it possible to consider vertical bit slices in sequential order, but this notion does not guarantee invertibility (for example, the mapping $x \to x + x^2$ $(mod\ 2^n)$ is a T-function but it maps both -1 and 0 to 0). To guarantee invertibility, we combine the notions of Feistel networks and triangulation in the following way:

Definition 4. *A* **triangular Feistel Network (TFN)** *is any mapping over bit matrices described by:*

$$\begin{aligned}&(p_{\star,0}, p_{\star,1}, \ldots, p_{\star,n-1}) \to \\ &(g_0(p_{\star,0}), g_1(p_{\star,1}; p_{\star,0}), \ldots, g_{n-1}(p_{\star,n-1}; p_{\star,0}, p_{\star,1}, \ldots, p_{\star,n-2})),\end{aligned} \tag{4}$$

in which g_0 is an invertible function and the g_i's for $i > 0$ are PIF's.

In fact, it is easy to prove that *any* T-function which is invertible is a TFN, and thus this is the most general construction of invertible T-functions. We prove this claim for bivariate mappings:

Lemma 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be finite sets, and let $\phi : \mathcal{A}\times\mathcal{B} \to \mathcal{A}\times\mathcal{B}$ be an invertible mapping of the form: $(a, b)\to(f(a), g(b, a))$. Then f is invertible and $g(b; a)$ is a PIF.*

Proof. Suppose that f is not invertible, so $|\{f(a)|a \in \mathcal{A}\}|$ is strictly smaller than $|\mathcal{A}|$. Then $|\{(f(a), \phi(a, b))|a \in \mathcal{A}, b \in \mathcal{B}\}| < |\mathcal{A}|\cdot|\mathcal{B}|$, which contradicts the invertibility of ϕ. If g is not a PIF, then there exists a, b_1 and b_2, s.t. $g(b_1; a) = g(b_2; a)$. Consequently, ϕ maps (a, b_1) and (a, b_2) to the same pair of values.

5 Testing the Invertibility of Parametric Functions

In this section we consider the related issues of how to construct invertible mappings from primitive functions, and how to test the invertibility of a given mapping of this type. The general problem is quite difficult, but in practice we will be interested mostly in univariate or bivariate mappings which are defined by a small number of machine instructions, and in such cases our techniques seem to work very efficiently.

Given such a mapping, we would like to test whether it is a TFN. We first have to test whether the mapping of the least significant bit slice is invertible. The simplest technique is to try the 2^m possible combinations of the least significant bits of the m variables, which is trivial for $m \leq 20$. Next we have to test whether for each $i > 0$, the parametric mapping $g_i(p_{\star,i}; p_{\star,0}, p_{\star,1}, \ldots, p_{\star,i-1})$ is a PIF. This cannot be tested by exhaustive search, since the number of cases we have to consider grows exponentially with i. Instead, we ignore the actual values of additive parameters when we evaluate the primitive functions by using the following recursive rules:

Lemma 2. *If $\circ$ is a boolean operation, a, b are words, and $i > 0$, then*

$$[a \circ b]_0 = [a]_0 \circ [b]_0 \tag{5}$$
$$[a \circ b]_i = [a]_i \circ [b]_i \tag{6}$$
$$[a + b]_0 = [a]_0 \oplus [b]_0 \tag{7}$$
$$[a + b]_i = [a]_i \oplus [b]_i \oplus \alpha \tag{8}$$
$$[a - b]_0 = [a]_0 \oplus [b]_0 \tag{9}$$
$$[a - b]_i = [a]_i \oplus [b]_i \oplus \alpha \tag{10}$$
$$[a \cdot b]_0 = [a]_0 \, [b]_0 \tag{11}$$
$$[a \cdot b]_i = [a]_i \, [b]_0 \oplus [a]_0 \, [b]_i \oplus \alpha \tag{12}$$
$$\left[a^k\right]_0 = [a]_0 \;\; for\ any\ k > 0 \tag{13}$$
$$\left[a^k\right]_i = [a]_i \, [a]_0 \oplus \alpha \;\; for\ any\ odd\ k > 0 \tag{14}$$
$$\left[a^k\right]_i = \alpha \;\; for\ any\ even\ k > 0 \tag{15}$$

where the α's are parameters denoting carries from previous bit positions.

Proof. We only prove the last claim — all the other cases use similar ideas. Since k is even $t = k/2$ is an integer, and thus we can use the claim about $[a \cdot b]_i$ to get $\left[a^k\right]_i = [a^t \cdot a^t]_i = [a^t]_i [a^t]_0 \oplus [a^t]_0 [a^t]_i \oplus \alpha = \alpha$. Consequently, any bit position in a^k (except the least significant one) depends only on lower bit positions in a, whose combination is considered as an additive parameter α.

To use these rules, we consider each expression in the multivariate mapping as a tree with variables at the leaves, primitive functions at the internal nodes, and one of the outputs at the root. We then compute the i-th bit of the output by traversing the tree and evaluating its slice operations. This process is algorithmically similar to formal differentiation of algebraic expressions. To demonstrate the process, consider the following bivariate mapping from the introduction:

$$x' = x + 2(x \wedge y) \tag{16}$$
$$y' = (y + 3x^3) \oplus x \tag{17}$$

It is not clear, a-priori, whether this mapping is invertible. To test it, we first compute the i-th bit (for any $i > 0$) of the first output as a parametric function:

$$\begin{aligned}[x']_i &= [x]_i \oplus [2(x \wedge y)]_i \\ &= [x]_i \oplus [2]_i [x \wedge y]_0 \oplus [2]_0 [x \wedge y]_i \oplus \delta \\ &= [x]_i \oplus [2]_i \alpha \oplus 0 \oplus \delta = [x]_i \oplus \theta\end{aligned}$$

where δ, α and θ are parameters which can depend on the previous bit slices but not on the i-th bit of any one of the input variables. In a similar way, we can derive the parametric representation of the i-th bit (for any $i > 0$) of the second output as

$$\begin{aligned}[y']_i &= \left[y + 3x^3\right]_i \oplus [x]_i = [y]_i \oplus \left[3x^3\right]_i \oplus \gamma \oplus [x]_i \\ &= [y]_i \oplus [3]_i \left[x^3\right]_0 \oplus [3]_0 \left[x^3\right]_i \oplus \delta \oplus \gamma \oplus [x]_i \\ &= [y]_i \oplus \alpha \oplus [x]_i [x]_0 \oplus [x]_i \oplus \epsilon = [y]_i \oplus \beta [x]_i \oplus \tau\end{aligned}$$

It is now easy to see that the two parametric slice mappings:

$$\begin{aligned}[x']_i &= [x]_i \oplus \theta \\ [y']_i &= [y]_i \oplus \beta [x]_i \oplus \tau\end{aligned} \tag{18}$$

are invertible for any $i > 0$ and for any choice of the multiplicative parameter β and the additive parameters θ and τ, and thus this mapping is a PIF. In addition, it is easy to test that the mapping defined by the least significant slice is invertible, and thus the original bivariate transformation over n bit words is invertible.

As another example, let us reprove in our framework Rivest's characterization in Theorem 1 of permutation polynomials modulo 2^n. First we check the case of $i = 0$: Since all the powers of x are equal to x modulo 2, we can simplify $[P(x)]_0 = \left[a_0 + a_1 x + \cdots + a_d x^d\right]_0 = [a_0]_0 + [(a_1 + \cdots + a_d)]_0 [x]_0$, which is invertible modulo 2 iff $a_1 + \cdots + a_d$ is odd.

Next we have to check which polynomials have invertible slice mappings for $i > 0$. By using our recursive simplification technique we get the following parametric representations:

$$\left[a_1 x^1\right]_i = [a_1]_i [x]_0 \oplus [a_1]_0 [x]_i \oplus \alpha_1 = [a_1]_0 [x]_i \oplus \alpha_2$$

For higher even powers of x we get:

$$\left[a_k x^k\right]_i = [a_k]_i \left[x^k\right]_0 \oplus [a_k]_0 \left[x^k\right]_i \oplus \alpha_3 = \alpha_4$$

and for higher odd powers of x we get:

$$\left[a_k x^k\right]_i = [a_k]_i \left[x^k\right]_0 \oplus [a_k]_0 \left[x^k\right]_i \oplus \alpha_5$$

$$= [a_k]_i [x]_0 \oplus [a_k]_0 ([x]_0 [x]_i \oplus \alpha_6) \oplus \alpha_5 = [a_k]_0 [x]_0 [x]_i \oplus \alpha_7$$

Bringing it all together, $[P(x)]_i = [a_1]_0 [x]_i \oplus \bigoplus_{odd\ k \geq 3} [a_k]_0 [x]_0 [x]_i \oplus \alpha$. To make sure that it is a PIF, this slice mapping should be invertible for both values of $[x]_0$. For $[x]_0 = 0$ we get the condition $[a_1]_0 = 1$, and for $[x]_0 = 1$ we get the condition $a_1 \oplus a_3 \oplus a_5 \oplus \cdots = 1$. Together with the condition $a_1 \oplus a_2 \oplus a_3 \cdots = 1$ from slice $i = 0$ we get Rivest's characterization of permutation polynomials in an almost mechanical way. As a bonus, we can use exactly the same proof to get exactly the same characterization of invertible mappings of the form $P(x) = a_0 \oplus a_1 x \oplus \cdots \oplus a_d x^d$ (which could not be handled by Rivest's algebraic technique), since there is no difference between + and $\oplus$ in the parametrized slice mappings.

As demonstrated by these examples, the invertibility of many types of mappings can be reduced to the invertibility of systems of linear equations modulo 2 with additive and multiplicative parameters (even when the original mapping is highly nonlinear and combines arithmetic and boolean operations). If we have only additive parameters, this invertibility does not depend on the actual values of the parameters and can be decided by evaluating a single determinant. Multiplicative parameters are more problematic, as we have to test the invertibility of parametrized matrices (modulo 2) for all the possible 0/1 values of the parameters. Since the number of parameters is usually small, exhaustive search is feasible, but this test can have a one-sided error: If all the parametrized matrices are invertible then the original mapping is invertible, but in the other direction noninvertible matrices may occur only for combinations of parameter values which can never happen for actual inputs. Consequently, this test provides a sufficient but not a necessary condition for invertibility.

An interesting special case happens when the parametrized coefficient matrix is triangular with 1's along its diagonal. Such matrices are invertible modulo 2 regardless of how parameters occur in its off-diagonal entries. This is demonstrated in the system of equations 18, which is invertible for any choice of the multiplicative parameter β.

When we consider the dual problem of constructing invertible mappings, we should choose primitive operations whose slice mappings are linear systems of equations with either no multiplicative parameters or with a triangular collection of multiplicative parameters. Alternatively, we can start with any simple

combination of primitive functions and use our technique to diagnose its invertibility. A failed test can indicate how the mapping should be modified in order to make it invertible. As a typical example, consider the mapping $x \to x \oplus x^2$. Its i-th slice mapping for any $i > 0$ is the invertible $[x']_i = [x]_i \oplus \alpha$, but for $i = 0$ we get the noninvertible $[x']_0 = [x]_0 \oplus [x]_0 = 0$. To solve this problem, all we have to do is to tweak the first slice mapping without affecting the other slice mappings. The simplest way to do this is to "or" the constant 1 to x^2, which changes only the first slice mapping to the invertible $[x']_0 = [x]_0 \oplus 1$. This leads in a natural way to the interesting mapping $x \to x \oplus (x^2 \vee 1)$, which is invertible for any word size.

6 Cryptographic Applications

So far we have defined a large class of complex mappings which can use a mixture of standard arithmetic and boolean operations on arbitrarily long words. They can be implemented very efficiently in software, and their invertibility can be easily tested. In this section we describe one of their cryptographic applications.

Since the actual inversion of these invertible mappings is quite inefficient, we would like to use them in situations where they are only executed in the forward direction. A typical example of such an application is their iterated use on the state S of a pseudo random number generator (PRNG). The initial state S_0 of the generator is derived from the key k, and after each iteration the PRNG outputs some simple function $\mathcal{O}(S_i)$ of its current state (e.g., its most significant bit or byte). It is essential to use invertible update operations to avoid an incremental loss of state entropy, but the update function is never run backwards in this application.

Our goal is to choose a good state update function which uses the smallest possible number of primitive functions, in order to achieve the highest possible speed in software implementation. An exhaustive analysis of all the mappings defined by up to two primitive functions indicate that there are only 8 families of invertible mappings of this type: $x \cdot C'$, $x + C$, $x \oplus C$, $(x + C_1) \oplus C_2$, $x \cdot C' + C$, $x \cdot C' \oplus C$, $x \cdot C'' \oplus x$, $(x \oplus C) \cdot C'$, where C, C_1 and C_2 are arbitrary constants, C' is an arbitrary odd constant, and C'' is an arbitrary even constant. Unfortunately, all these families are useless, either because they have a simple linear structure, or because they have very short cycles. For example, the states defined by the mapping $x \oplus C$ just oscillate between two possible states, and thus the output of the PRNG is highly predictable. Consequently, we have to use at least 3 primitive operations to update the state of the PRNG.

A very desirable property of an invertible state update mapping in this application is that all the 2^n possible states should be connected by a single cycle of length 2^n. In the case of linear feedback shift registers, this is almost achieved by using a primitive feedback polynomial (which leaves only the all-zero state in a second cycle). In general, it is very difficult to determine the cycle structure of complex invertible mappings, but the possible cycle structures of T-functions are severely limited by the following observation:

Lemma 3. *Let $f_n(x) = f(x) \pmod{2^n}$ be an invertible T-function. Then for each cycle in f_{n-1} of length l, there are either two cycles of length l or one cycle of length $2l$ in f_n. Consequently, if f has any fixed point then for any $i > 0$ there are at least 2^{i+1} points that belong to cycles of length at most 2^i.*

For $n = 0$ the only possible forms of f_0 are $f_0(x) = x$ or $f_0(x) = x \oplus 1$ which have cycles of length 1 and 2. Consequently, the length of any cycle of an invertible T-function is a power of 2. Another trivial corollary is that we should only use functions which have no fixed points at all, since otherwise there are at least two fixed points, at least four points on cycles of length bounded by 2, etc.

Consider, for example, the well known RC6 function $f_n(x) = x(2x + 1) \pmod{2^n}$. Unfortunately, it has a fixed point $x = 0$, and thus it has several small cycles. In fact, its cycle structure is much worse than the general bound given above (for example, for $n = 3$, 4 out of the 8 possible inputs are fixed points). It is easy to show that for any n this f_n has exactly $2^{\frac{n}{2}}$ fixed points, exactly $2^{\frac{n}{2}-2}$ points on cycles of length 2, etc. This makes it very unsuitable for our PRNG application.

A particularly interesting family of mappings is $x \to x + (x^2 \vee C)$ for various constants C and word sizes n, for which we can prove:

Theorem 3. *The mapping $f(x) = x + (x^2 \vee C)$ over n bit words is invertible if and only if the least significant bit of C is 1. For $n \geq 3$ it is a permutation with a single cycle if and only if both the least significant bit and the third least significant bit of C are 1.*

Proof. The first claim is trivial. For any bit slice $i > 0$ $[f(x)]_i = [x]_i \oplus \alpha$ (where α is a parameter) is invertible, and thus the only possible problem is the least significant bit slice $i = 0$. There are two cases to consider: If $[C]_0 = 0$, $[f(x)]_0 = [x]_0 \oplus [x]_0 = 0$, whereas if $[C]_0 = 1$, $[f(x)]_0 = [x]_0 \oplus 1$. Consequently, the invertibility of $f(x)$ depends only on the least significant bit of the constant C.

The second claim is more complicated, and we prove it by induction on the word length n. For $n = 3$ we can show by simple enumeration that the function $f(x) = x + (x^2 \vee C) \pmod{2^3}$ has a single cycle if and only if $C = 5$ or $C = 7$. Note that this already implies that for any $n \geq 3$ $f(x)$ has more than one cycle whenever $C \notin \{5, 7\} \pmod 8$, and thus we only have to prove that the permutation has a single cycle whenever $C \in \{5, 7\} \pmod 8$ (i.e., whenever both the least significant bit and the third least significant bit of C are 1).

The length of any cycle of any T-permutation is of the form 2^k for some k. Our strategy is to show that $\forall n \exists x : \left[f^{(2^{n-1})}(x)\right]_{n-1} = [x]_{n-1} \oplus 1$, since this implies the existence of some cycle of length larger than 2^{n-1}, and the only possible cycle length is then the full 2^n.

Assume by induction that f has only one cycle for $n-1$. Consider the sequence x_0, $x_1 = f(x_0) = x_0 + (x_0^2 \vee C)$, $x_2 = f^{(2)}(x_0) = f(x_1) = x_1 + (x_1^2 \vee C) = x_0 + (x_0^2 \vee C) + (x_1^2 \vee C)$, …, $x_{2^{n-1}} = f^{(2^{n-1})}(x) = x_0 + \sum_{i=0}^{2^{n-1}-1}(x_i^2 \vee C)$. From the assumption that there is only one cycle for $n - 1$, we know that $\{x_i \bmod 2^{n-1}\}_{i=0}^{2^{n-1}-1}$ is just a permutation of $\{i\}_{i=0}^{2^{n-1}-1}$. Since $\left[x^2\right]_{n-1}$ does not depend

on $[x]_{n-1}$ the set $\{(x_i^2) \bmod 2^n\}_{i=0}^{2^{n-1}-1}$ is the same as $\{((x_i \bmod 2^{n-1})^2) \bmod 2^n\}_{i=0}^{2^{n-1}-1}$. So $f^{(2^{n-1})}(0) \bmod 2^n = \sum_{j=0}^{2^{n-1}-1}(j^2 \vee C) \bmod 2^n$. An expression $x \vee 2^i$ is equal either to x if $[x]_i = 1$ or to $x + 2^i$ otherwise. Similarly $j^2 \vee C = j^2 + \sum_{i:[C]_i=1 \wedge [j^2]_i=1} 2^i$ and $\sum_{j=0}^{2^{n-1}-1}(j^2 \vee C) = 2^{n-1}(\sum_{i:[C]_i=1} 2^j z_j)$. Here z_i denotes the probability of '0' in bit number i of x^2 for random x, i.e., $\frac{\#\{[x^2]_i=0\}_{x=0}^{2^i-1}}{2^i}$. It is easy to show that $z_0 = \frac{1}{2}$, $z_1 = 1$ and $\forall i > 1 : z_i = \frac{1}{2}(1 + 2^{-\lfloor \frac{i}{2} \rfloor})$. [1] The formula for the sum of squares is $\sum_{j=0}^{k} j^2 = \frac{k^3}{3} + \frac{k^2}{2} + \frac{k}{6}$, so $\sigma = f^{(2^{n-1})}(0) = (\frac{2^{3n}}{3} + \frac{2^{2n}}{2} + \frac{2^n}{6}) - 2^{2n} + 2^{n-1}(\sum_{j:[C]_j=1} 2^j z_j)$ and we want to prove that $[\sigma]_{n-1} = 1$. This is determined by the least significant bit of $\frac{\sigma}{2^{n-1}} = \frac{1}{3}(2^{2n+1} + 2^n + 1) - 2^{n+1} + \sum_{j:[C]_j=1} 2^j z_j$. In binary $\frac{1}{3} = 0.(10)_2$ and $\frac{2^k}{3}$ is either $1010\ldots10.(10)_2$ or $1010\ldots1.(01)_2$, so the infinite fraction $\frac{1}{3}(2^{2n+1} + 2^n)$ has either the form $0.(10)_2$ (if i is odd) or the form $1.(1)_2$ (if i is even). In both cases it represents a number which is equal to '0' modulo 2. In addition $2^j z_j = 0 \pmod 2$ for all j except '0' and '2', for which the values are $\frac{1}{2}$ and 3, respectively. Finally $\frac{\sigma}{2^{n-1}} \pmod 2 = \frac{1}{3} + \frac{1}{2} + 3 \pmod 2 = 1$ and thus for $x = 0$, $\left[f^{(2^{n-1})}(0)\right]_{n-1} = 1 = [0]_{n-1} \oplus 1$ which completes the proof.

In particular, $x + (x^2 \vee 1)$ is invertible but has multiple cycles of various sizes, whereas $x + (x^2 \vee 5)$ has only one cycle of length 2^n and thus it is a better choice for a PRNG application.

V. S. Anashin [1] published a somewhat related analysis of the cycle structure of functions defined over p-adic numbers. He used notions of ergodicity and measure preservation in order to obtain topological characterization of such mappings. In particular, he proved a theorem which could be translated into our notation in the following way: if $c, r \neq 0 \pmod p$ then $f(x) = d + cx + pv(x) \pmod{p^n}$ is invertible and $g(x) = c + rx + p(v(x+1) - v(x)) \pmod{p^n}$ has only one cycle. For example, the invertibility of the RC6 function $x(2x+1)$ could be shown by this theorem, but neither the invertibility nor the cycle structure of $x + (x^2 \vee 5)$ could be determined by his techniques.

The pure PRNG's produced by invertible T-functions are not cryptographically secure by themselves, but they can serve as excellent building blocks in software based generators (in the same way that linear feedback shift registers are insecure as stand-alone designs, but they can serve as excellent components in hardware based generators). They can have provably long cycles, and unlike the case of LFSR's, there is no need to worry about weak keys that fill the shift

[1] For $i \leq 2$ this could be checked by direct calculation. Every n-bit number x' could be represented as either $2x$ or $2x+1$, where x has $n-1$ bits. In the first case $\Pr[[x'^2]_i = 0 \,|\, [x']_0 = 0] = z_{i-2}$, and in the second $\Pr[[x'^2]_i = 0 \,|\, [x']_0 = 1] = \Pr[[4x^2 + 4x + 1]_i = 0] = \Pr[[x^2]_{i-2} \oplus [x]_{i-2} \oplus \alpha = 0] = \frac{1}{2}$ (where α is a carry), since $[x]_{i-2}$ is independent of other bits (again note that $[x^2]_i$ does not depend on $[x]_i$ for $i > 0$) and equally likely to be 0 and 1. Combining both cases we get an inductive proof of the claim.

register with zeroes and get stuck in a fixedpoint. Even the simplest generators can have excellent statistical properties. For example, we used the statistical test suite [8] which was developed for the AES candidates [10], and we found that the sequence of upper halves (top 32 bits) of the 64 bit numbers defined by the iterated use of $x_{i+1} = x_i + (x_i^2 \vee 5) \pmod{2^{64}}$ is statistically random with significance level $\alpha = 0.01$ (which is better than for some of the AES candidates themselves!). Since each iteration requires only 3 machine instructions, we can use it as an exceptionally fast source of (weak) pseudorandomness.

Note that unlike the case of LFSR's, when we iterate a T-function there is no propagation of information from left to right in the state word. To overcome this problem, we can alternately apply a T-function and a cyclic rotation to the state. The combined mapping is clearly invertible, has both left-to-right and right-to-left information propagation, but it is very difficult to analyse its cycle structure. Consequently, such combinations have to be studied very carefully before they can be adopted in new cryptographic schemes, and in particular one should consider the linear and differential properties of these mappings, the best way to combine them into stronger schemes, the best attacks against such combinations, how many output bits can be extracted from the current state in each iteration, etc.

7 Concluding Remarks

This paper proposes a new cryptographic building block which mixes boolean and arithmetic operations, and analyses some of its cryptographic properties. In particular, we use the multiplication operation (which on many modern microprocessors take about the same time as addition) to thoroughly mix the input bits in a nonlinear way and to enhance the statistical properties of the mapping. In addition, we use the boolean operations to mask the algebraic weaknesses of low degree polynomial mappings, and to control their cycle structures. The resultant mappings have very compact and extremely fast software implementations, and thus they could replace large S-boxes or LFSR's in many software based cryptographic schemes. We believe that as we move from 32 to 64 (and later to 128) bit processors, such wide-word mappings over mixed algebraic structures will become an increasingly natural and attractive alternative to traditional bit and byte oriented designs.

References

1. V. S. Anashin, "Uniformly distributed sequences over p-adic integers", Proceedings of the Int'l Conference on Number Theoretic and Algebraic Methods in Computer Science (A. J. van der Poorten, I. Shparlinsky and H. G. Zimmer, eds.), World Scientific, 1995.
2. E. Biham, "A Fast New DES Implementation in Software", Fast Software Encryption Workshop, 1997
3. H. Feistel, "Cryptography and Computer Privacy," Scientific American, v. 228, n. 5, May 1973, pp. 15–23.

4. V. Furman, "Differential Cryptanalysis of Nimbus", Fast Software Encryption Workshop, 2001
5. H. Lipmaa, S. Moriai, "Efficient Algorithms for Computing Differential Properties of Addition", 2001. Available from `http://citeseer.nj.nec.com/lipmaa01efficient.html`
6. A. W. Machado, "The nimbus cipher: A proposal for NESSIE", NESSIE Proposal, 2000.
7. National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.
8. The NIST Statistical Test Suite. Available from `http://csrc.nist.gov/rng/rng2.html`
9. J. Daemen, V. Rijmen, "AES Proposal: Rijndael", version 2, 1999
10. Randomness Testing of the AES Candidate Algorithms. Available from `http://csrc.nist.gov/encryption/aes/round1/r1-rand.pdf`
11. R. Rivest, M. Robshaw, R. Sidney, and Y. L. Yin, "The RC6 block cipher". Available from `http://www.rsa.com/rsalabs/rc6/`
12. R. Rivest, "Permutation Polynomials Modulo 2^ω", 1999.
13. B. Schneier and J. Kelsey, "Unbalanced Feistel Networks and Block Cipher Design", in Proceedings of the Third International Workshop on Fast Software Encryption, Cambridge, UK, February 1996, Springer, LNCS 1039, pp.121–144.
14. A. Shamir, "Efficient Signature Schemes Based on Birational Permutations", in Proceedings of CRYPTO 93, LNCS 773, 1–12.