CRYPTOGRAPHY: MATH TRUMPS BLACK MAGIC

PAUL CARR

1. Introduction

Cryptography is the process of turning large secrets into small ones. The basic idea is to use a short, fixed-length **key** to turn an arbitrarily large message into a similarly-sized chunk of unintelligible data in such a way that by possessing both the gibberish and the key, one can reconstruct the original message. The problem of moving a potentially very large message between two people securely is then reduced to moving a small key. In some algorithms, the encrypting and decrypting key are different, and this problem is removed entirely. These special methods are called **asymmetric** or **public** key algorithms.

Many of these algorithms, not surprisingly, are based on mathematical concepts. Elliptic curves, discrete logarithms, and products of large prime numbers are the foundations of several. These algorithms, particularly RSA, are regularly invoked in number theory, abstract algebra, and similar classes as real-world applications of particular concepts. The downside of the asymmetric key algorithms, however, is their relative slowness compared to other cryptographic algorithms. They also typically require significantly larger keys.

The workhorses of cryptography are the symmetric key algorithms, which generally make use of only a handful of very basic, very fast, computer functions, particularly **exclusive or** (**xor**), which is bitwise addition modulo 2, left and right shifts (multiplication or division by powers of two, with truncation), and small lookup table function representations. These algorithms are not based on simple mathematical results, but instead attempt to use these small, rapid operations in a large number of iterations to obfuscate the message. With the exception of lookup tables, called **substitution boxes** (or **S-boxes**, for short), all of these operations are linear, and thus a great deal of any algorithm's security is predicated on choosing good S-boxes. In general this has been done by generating random lookup tables, and throwing out any that are vulnerable to known attack techniques. More recently, however, mathematical rigor has been brought to this process, and the most significant result has been the selection of an algorithm called Rijndael, designed in this way, as the new US government Advanced Encryption Standard (AES), primarily based on their design methodology.

Date: February 27, 2005.

We will attempt to develop the tools and concepts used in the quantification of design goals for the nonlinear S-box function, consider several satisfactory S-boxes, and then look at what, specifically, the creators of Rijndael decided upon.

2. Anatomy of a Cryptographic Algorithm

We must outline some terminology and jargon before proceeding with the good stuff, so settle in.

Plaintext, **ciphertext**, and **key** are the three variables that our functions are going to be acting on, and inhabit the domains and ranges in various arrangements. When we are interested in making a message or other chunk of data unreadable by malicious creatures, we chop it up into **blocks** of plaintext: binary strings of a particular length. Then we secure each block individually, using a key (also a binary string). Once a block of plaintext is secured (or **encrypted**), it becomes a block of ciphertext (it is *en*ciphered, or unreadable). Possession of a key grants the ability to *de*cipher (**decrypt**) the ciphertext and recover the plaintext.

A **cipher** consists of a description of an encrypting function, a decrypting function, and all the options that determine various characteristics, particularly the block size and the key size. For instance, the DES cipher can be run with a key length of either 40 or 56 bits. The shorter key makes encrypting and decrypting faster, but easier to break. Note that ciphers do not have to act strictly on blocks- those that do are called **block ciphers**. Once the options are set, we end up with a function for encrypting and another for decrypting.

So now we move to what makes up an **encrypting function** or a **decrypting function**. Such a function maps two inputs to one output. The inputs to the encrypting function F_E are a block of plaintext x and a key k, and the output is a block of ciphertext y. The inputs to the decrypting function F_D are y and k, and the output is x. Thus

$$F_D(F_E(x,k),k) = x.$$

So, for a fixed value of k, F_D and F_E are inverses of one another. We call these functions **keyed**.

F_D and F_E are fairly elaborate and contain many individual components which are also functions. The main units inside these functions are called **rounds**. The general structure of a block cipher is some kind of initial function, then a **round function** applied a number of times iteratively, and finally some kind of concluding function. This design choice is based to some extent on efficiency constraints, but it also clearly has ramifications for security. The round function also takes a block of input and a key and generates a block of output. The key for each round is typically different, and derived from the main algorithm key in such a way that knowledge of any number of round keys does not disclose any information about any other

round key. It may be said that they are (or can be closely approximated by) independent random variables, from a statistics standpoint.

A round function also contains individual component functions. Typically these serve two purposes: **confusion** and **diffusion**. The goal of confusion is to make it difficult to determine an input to a round function, given an output. This is usually carried out by a single nonlinear invertible function, the S-box. The goal of diffusion is to make each output bit depend on as many input bits as possible. This is carried out by a series of linear functions.

It is clear, then, that the decrypting function will be built up in a similarly piecewise fashion, involving the inverse of each of the pieces of the encrypting function.

3. Building a Model

Now we need a model to represent this situation. Clearly the dominant behavior is that of functions acting on spaces. What spaces, then? Plaintexts, ciphertexts, and keys are all strings of binary digits of specific lengths. Integers, then. But not all integers, only those on the interval $[0, 2^n - 1]$, where n is the number of bits allowed in a block, or the key length, depending on which object we're talking about. We also need operations on these numbers. We can introduce addition, and make it closed under it by taking all sums modulo 2^n, and thus create a group. However, the most common interaction between strings of bits in cryptography is not plus, but exclusive or (xor), which is a bitwise operation, adding each of the strings together, bit by bit, modulo 2. Using xor as our $+$, we retain group structure. Xor is also linear. This bitwise addition suggests a vector space, specifically an n-dimensional vector space over $\mathbb{Z}_2 = \{0, 1\}$. This gives us a strong handle on all *linear* functions on blocks of plaintext, ciphertext, and keys, in the form of transformation matrices.

Most of the functions in a cipher are linear, but not all. We need other operations. Since we're already playing with groups, it is natural to extend our gaze to rings, or fields. In fact, Galois Fields offer a very nice representation for this situation. $\mathrm{GF}(2^n)$ is a finite field whose contents are objects of the form $\{(a_0, a_1, ..., a_{n-1}) \ : \ a_i \in \mathbb{Z}_2\}$, where the $+$ operation is carried out independently on each variable, mod 2. This describes n-bit binary strings under xor exactly, and grants us the additional tool of a multiplication operator which offers the possibility of nonlinearity.

So what *is* multiplication in $\mathrm{GF}(2^n)$? We change representations from vectors to polynomials: $(a_0, a_1, ..., a_{n-1})$ becomes

$$a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1}.$$

To multiply two polynomials, we must first choose an irreducible polynomial p of degree n to define the multiplication operation on our field. We then define the product of $a = (a_0, a_1, ..., a_{n-1})$ and $b = (b_0, b_1, ..., b_{n-1})$ to be

$$ab \pmod{p}.$$

That is, find polynomials q and c, with the degree of c less than n, such that $ab = qp + c$ (using, for instance, the Euclidean algorithm). Then, the product of a and b in $\mathrm{GF}(2^n)$ is defined to be c.

For each positive integer n there are one or more irreducible polynomials to choose from, but the choice is not particularly important for our purposes- the fields determined by each option are all isomorphic.

We can use this multiplication operation to define functions on $\mathrm{GF}(2^n)$ that are nonlinear, in various ways. Of particular interest for our purposes are power functions, such as $f(a) = a^3 = a(x) \cdot a(x) \cdot a(x)$.

It should be noted in passing that polynomial addition behaves identically to our previous description of vector addition, which we already decided behaved identically to the xor operation, on elements of $\mathrm{GF}(2^n)$. So, the polynomial description of $\mathrm{GF}(2^n)$ is sufficient to describe everything we're working with. However, when dealing with linear functions we will drop back to vector spaces sometimes, when it makes for a clearer descriptive framework.

4. Functions on $\mathrm{GF}(2^n)$

So let's figure out some things about functions on $\mathrm{GF}(2^n)$. It is convenient to talk about **boolean functions** first. These are functions from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2)$, that is, functions that output either zero or one. So, among boolean functions, what categories are interesting? Linear, certainly. A linear boolean function can be defined completely by a vector in $\mathrm{GF}(2^n)$: $f(x) = t^\mathsf{T} x = t_0 x_0 + ... + t_{n-1} t_{n-a}$. It can be looked at as a matrix transformation from n dimensions to 1 dimension, or as a dot product. From linear we can extend naturally to affine: just add a zero or a one to the result. So if a linear boolean function can be defined by an n-dimensional vector, an affine boolean function can be defined by an $(n+1)$-dimensional vector. Let's formalize this:

Definition 4.1. A function f from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2)$ is called a **boolean function**. If f can be expressed as $f(x) = t^\mathsf{T} x$ for some $t \in \mathrm{GF}(2^n)$, then it is called a **linear boolean function**. If f can be expressed as $f(x) = t^\mathsf{T} x + b$ for $t \in \mathrm{GF}(2^n)$, $b \in \mathrm{GF}(2)$, it is called an **affine boolean function**. The set of all linear boolean functions on $\mathrm{GF}(2^n)$ will be denoted $\mathcal{L}_n$, and the set of all affine boolean functions on $\mathrm{GF}(2^n)$ will be denoted $\mathcal{A}_n$.

Note that $\mathcal{A}_n$ contains exactly the linear boolean functions and their complements. This definition tells us, among other things, that there are exactly 2^n linear boolean functions and 2^{n+1} affine boolean functions on $\mathrm{GF}(2^n)$. As any boolean function can be represented by a lookup table with 2^n entries, defining the output for each possible input, there are a total of 2^{2^n} possible boolean functions on $\mathrm{GF}(2^n)$.

At first glance it seems unreasonably restrictive to work with just boolean functions, when our goal is to deal with permutations and other

$\mathrm{GF}(2^n) \mapsto \mathrm{GF}(2^m)$ mappings. However, any such function can be represented as a vector of m component functions:

$$F(x) = (f_1, f_2, ..., f_m)$$

where each f_i is a boolean function on $\mathrm{GF}(2^n)$. Whenever possible, we will attempt to cast properties of these general functions in terms of properties of boolean functions. Thus it behooves us to become familiar with these simpler functions before moving on to the general case.

It is useful to note here that linearity on $\mathrm{GF}(2^n)$ is less work to prove than in general, as scalar multiplication is a nonissue, since the only scalars are 1 (identity, so linear functions stay linear) and 0 (makes any function 0, which is as linear as it gets). Thus, to prove a function on $\mathrm{GF}(2^n)$ linear, we need only show the additivity property.

If we choose a boolean function at random (by generating a random lookup table for an S-box, for instance), the odds of it being affine are very small, and approach zero as n becomes large. Since almost all functions are nonlinear, it is reasonable to wish to distinguish between nonlinear functions in some way- to be able to say that one function is *more* nonlinear than another. To do this we need to talk about distances.

Definition 4.2. Given f, g boolean functions on $\mathrm{GF}(2^n)$, the distance between f and g is defined to be:

$$d(f, g) = \#\big\{x \in \mathrm{GF}(2^n) : f(x) \neq g(x)\big\}$$

where # denotes cardinality.

This is also called the **hamming distance** between boolean functions. It is directly related to the hamming distance between vectors, as a boolean function can be represented by a 2^n-dimensional vector over $\mathrm{GF}(2)$, with element x in the vector equal to the boolean function evaluated at x. This distance is thus also the "taxicab" (or L^1) metric.

Now that we have a distance, we can take a stab at measuring how nonlinear a boolean function is.

Definition 4.3. Given boolean function f and set G of boolean functions, all on $\mathrm{GF}(2^n)$, the distance between f and G is defined to be:

$$d(f, G) = \min\{d(f, g) : g \in G\}.$$

Of particular interest for our purposes is $d(f, \mathcal{A}_n)$, the distance of a function f from the affine functions. We will tentatively define this as the **nonlinearity** of f. Note that the terms linear and affine are used in a less than distinct way. This is the convention, however, so we will abide by it. The word "nonaffinity" has a somewhat unpleasant sound in any event, so perhaps it is for the best. In almost all cases it is the affine functions that we will be concerned with.

5. Convolutions and the Walsh Transform

Now we will define and consider some tools that will be useful later.

Definition 5.1. The **convolution** $f * g$, of functions f and g on $\mathrm{GF}(2^n)$ (not necessarily, but usually, boolean) is defined to be:

$$(f * g)(a) = \sum_{x \in \mathrm{GF}(2^n)} f(x)g(x + a).$$

Definition 5.2. Let f be a mapping from $\mathrm{GF}(2^n)$ to {1,-1}. Then the **Walsh transform** of f is an invertible mapping into the real-valued functions. It is denoted as either $\mathsf{w}(f)$ or $\hat{\mathrm{F}}(t)$, depending on context, and is defined to be:

$$(\mathsf{w}(f))(t) = \hat{\mathrm{F}}(t) = \sum_{x \in \mathrm{GF}(2^n)} f(x)(-1)^{t^{\mathsf{T}}x}, \quad t \in \mathrm{GF}(2^n).$$

The inverse mapping (denoted $\mathsf{w}^{-1}(\hat{\mathrm{F}})$) is:

$$f(x) = 2^{-n} \sum_{t \in \mathrm{GF}(2^n)} \hat{\mathrm{F}}(t)(-1)^{t^{\mathsf{T}}x}, \quad x \in \mathrm{GF}(2^n).$$

Note that the Walsh transform acts on functions that are not boolean by our definition. This is for clarity in notation in the following results, and we will extend the Walsh transform to the boolean functions in a natural way shortly.

Now we will state and prove two theorems involving convolutions and the Walsh transform.

Theorem 5.3. (Convolution Theorem) *For f, g functions from $GF(2^n)$ to $\{1, -1\}$:*

$$\mathsf{w}(f * g) = \mathsf{w}(f)\mathsf{w}(g).$$

Proof. Let $\hat{\mathrm{F}} = \mathsf{w}(f),\ \hat{\mathrm{G}} = \mathsf{w}(g),\ \hat{\mathrm{H}} = \mathsf{w}(f+g)$. Then

$$\begin{aligned}
\hat{\mathrm{H}}(t) &= \sum_{a\in \mathrm{GF}(2^n)} (f*g)(a)(-1)^{t^{\mathsf{T}}a} \\
&= \sum_{a\in \mathrm{GF}(2^n)} \left(\sum_{x\in \mathrm{GF}(2^n)} f(x)g(x+a) \right) (-1)^{t^{\mathsf{T}}a} \\
&= \sum_{a\in \mathrm{GF}(2^n)} \left(\sum_{x\in \mathrm{GF}(2^n)} f(x)g(x+a)(-1)^{t^{\mathsf{T}}a} \right) \\
&\quad \text{(substitute } y = x+a\text{)} \\
&= \sum_{y\in \mathrm{GF}(2^n)} \left(\sum_{x\in \mathrm{GF}(2^n)} f(x)g(y)(-1)^{t^{\mathsf{T}}(x+y)} \right) \\
&= \sum_{y\in \mathrm{GF}(2^n)} \left(\sum_{x\in \mathrm{GF}(2^n)} f(x)(-1)^{t^{\mathsf{T}}x} g(y)(-1)^{t^{\mathsf{T}}y} \right) \\
&= \left(\sum_{x\in \mathrm{GF}(2^n)} f(x)(-1)^{t^{\mathsf{T}}x} \right) \left(\sum_{y\in \mathrm{GF}(2^n)} g(y)(-1)^{t^{\mathsf{T}}y} \right) \\
&= \hat{\mathrm{F}}(t)\hat{\mathrm{G}}(t).
\end{aligned}$$

□

That is, the Walsh transform of a convolution is the product of the Walsh transforms of the convoluted functions. Now, one more general result about Walsh transforms:

Theorem 5.4. (Parseval's Theorem) *For boolean function f on $GF(2^n)$, and $\hat{F} = \mathsf{w}(f)$:*

$$2^n \sum_{x\in GF(2^n)} [f(x)]^2 = \sum_{t\in GF(2^n)} \left[\hat{F}(t)\right]^2.$$

Proof. Note first that

$$\begin{aligned}
(f*f)(a) &= \mathsf{w}^{-1}\left((\mathsf{w}(f*f))(a)\right) \\
&= 2^{-n} \sum_{t\in \mathrm{GF}(2^n)} \left((\mathsf{w}(f*f))(a)\right)(t)(-1)^{t^{\mathsf{T}}a}.
\end{aligned}$$

Using this fact, we can achieve the desired result:

$$\begin{aligned}
&2^n \sum_{x\in \mathrm{GF}(2^n)} [f(x)]^2 \\
&= 2^n \sum_{x\in \mathrm{GF}(2^n)} f(x)f(x) \\
&= 2^n (f * f)(0) \\
&= (2^n)(2^{-n}) \sum_{t\in \mathrm{GF}(2^n)} (\mathsf{w}(f * f))(t) \\
&= \sum_{t\in \mathrm{GF}(2^n)} [\hat{\mathrm{F}}(t)]^2 \quad \text{(by the Convolution Theorem).}
\end{aligned}$$

□

Now we will extend the Walsh transform to apply to what we're interested in: boolean functions.

Definition 5.5. Let f be a boolean function on $\mathrm{GF}(2^n)$. Then the Walsh transform of f is defined to be:

$$(\mathsf{w}(f))(t) = \hat{\mathrm{F}}(t) = \sum_{x\in \mathrm{GF}(2^n)} (-1)^{f(x)}(-1)^{t^{\mathsf{T}}x}.$$

We can now use the Walsh transform to give another way to calculate the nonlinearity of boolean functions. The following reasoning is taken from [12].

Proposition 5.6. *For all boolean functions f on $GF(2^n)$,*

$$d(f, \mathcal{A}_n) = 2^{n-1} - \frac{1}{2} \max_{t\in GF(2^n)} |\hat{F}(t)|.$$

Proof. Let $\hat{\mathrm{F}}$ be the Walsh transform of f. Then

$$\begin{aligned}
\hat{\mathrm{F}}(t) &= \sum_{x\in \mathrm{GF}(2^n)} (-1)^{f(x)}(-1)^{t^{\mathsf{T}}a} \\
&= \sum_{x\in \mathrm{GF}(2^n)} (-1)^{f(x)+t^{\mathsf{T}}a} \\
&= \sum_{\substack{x\in \mathrm{GF}(2^n) \\ f(x)=t^{\mathsf{T}}x}} (1) + \sum_{\substack{x\in \mathrm{GF}(2^n) \\ f(x)\neq t^{\mathsf{T}}x}} (-1) \\
&= \#\{x \in \mathrm{GF}(2^n) : f(x) = t^{\mathsf{T}}a\} - \#\{x \in \mathrm{GF}(2^n) : f(x) \neq t^{\mathsf{T}}a\} \\
&= (2^n - d(f, t^{\mathsf{T}}a)) - (d(f, t^{\mathsf{T}}a)) \\
&= 2^n - 2d(f, t^{\mathsf{T}}a).
\end{aligned}$$

By minor rearrangement, we get something useful:

$$d(f, t^{\mathsf{T}}x) = 2^{n-1} - \frac{1}{2}\hat{\mathrm{F}}(t). \tag{1}$$

Now we have the distance of f from $\mathcal{L}_n$. The other half of $\mathcal{A}_n$ contains the complements of each function in $\mathcal{L}_n$. For an affine function $t^{\mathsf{T}}x + 1$,

$$d(f, t^{\mathsf{T}}x + 1) = 2^n - d(f, t^{\mathsf{T}}x).$$

So, through some more minor manipulation, we arrive at:

$$d(f, t^{\mathsf{T}}x + 1) = 2^{n-1} + \frac{1}{2}\hat{\mathrm{F}}(t). \tag{2}$$

So, now we have a formula for the distance of f from every function in $\mathcal{A}_n$. By combining (1) and (2), we arrive at:

$$\begin{aligned} d(f, \mathcal{A}_n) &= \min\{\min\{d(f, t^{\mathsf{T}}x), d(f, t^{\mathsf{T}}x + 1)\} : t \in \mathrm{GF}(2^n)\} \\ &= \min\{2^{n-1} - \frac{1}{2}|\hat{\mathrm{F}}(t)| : t \in \mathrm{GF}(2^n)\} \\ &= 2^{n-1} - \frac{1}{2}\max\{|\hat{\mathrm{F}}(t)| : t \in \mathrm{GF}(2^n)\}. \end{aligned}$$

So, the final result, after all that work, is what we want:

$$d(f, \mathcal{A}_n) = 2^{n-1} - \frac{1}{2}\max_{t \in \mathrm{GF}(2^n)} |\hat{\mathrm{F}}(t)|. \tag{3}$$

□

Finally, let us consider a particular set of boolean functions that we will prove achieve maximum distance from $\mathcal{A}_n$. Let the set P_n contain all boolean functions on $\mathrm{GF}(2^n)$ with the property that $(f * f)(a) = 0$ for all nonzero a. That is, $f(x)$ and $f(x + a)$ differ at exactly half of their inputs. Another way to state this is:

$$\sum_{x \in \mathrm{GF}(2^n)} (-1)^{f(x)+f(x+a)} = 0 \quad \forall a \in \mathrm{GF}(2^n),\ a \neq 0.$$

Note that we are not making any claim of existence here: P_n may be empty, and in fact it *is* empty for all odd n. Refer to Theorem 3.3B in [12] for an explicit construction of some elements of P_n for all even n. Now for why we're interested in this set:

Theorem 5.7. *For all f in P_n,*

$$\mathcal{N}(f) = 2^{n-1} - 2^{\frac{n}{2}-1}.$$

Furthermore, this is the maximum possible nonlinearity for any boolean function on GF(2^n).

Proof. By the Convolution Theorem:

$$\mathsf{w}(f * f) = [\mathsf{w}(f)]^2.$$

Let's fiddle with the left side of this equation:

$$\begin{aligned}
(\mathsf{w}(f * f))(t) &= \sum_{a \in \mathrm{GF}(2^n)} (f * f)(a)(-1)^{t^{\mathsf{T}} a} \\
&= (f * f)(0)(-1)^{t^{\mathsf{T}} 0} + \sum_{\substack{a \in \mathrm{GF}(2^n) \\ a \neq 0}} 0(-1)^{t^{\mathsf{T}} a} \\
&= (f * f)(0) \\
&= \sum_{x \in \mathrm{GF}(2^n)} f(x) f(x) \\
&= \sum_{x \in \mathrm{GF}(2^n)} (\pm 1)^2 \\
&= 2^n.
\end{aligned}$$

Thus $[\hat{\mathrm{F}}(t)]^2 = 2^n$ for all $t \in \mathrm{GF}(2^n)$.

This is a very high nonlinearity, close to the trivial upper bound of 2^{n-1}, but is it maximal? Suppose that boolean function f is **not** in P_n. Then by Parseval's Theorem:

$$\sum_{t \in \mathrm{GF}(2^n)} [\hat{\mathrm{F}}(t)]^2 = 2^n \sum_{x \in \mathrm{GF}(2^n)} [f(x)]^2 = 2^{2n}.$$

Since $|\hat{\mathrm{F}}(t)| \neq 2^{\frac{n}{2}}$ for at least one $t \in \mathrm{GF}(2^n)$, there must exist a t with $|\hat{\mathrm{F}}(t)| > 2^{\frac{n}{2}}$. This implies that

$$d(f, \mathcal{A}_n) < 2^{n-1} - 2^{\frac{n}{2}-1}.$$

Thus f is "less" nonlinear than the elements of P_n.

□

What this means is that the set P_n, if nonempty, is the set of maximally nonlinear boolean functions on $\mathrm{GF}(2^n)$, by our measure of nonlinearity. These functions are called **perfectly nonlinear** and are the same as the **bent** functions in combinatorics.

6. The Field Trace

Another tool that will be useful in manipulating functions on $\mathrm{GF}(2^n)$ is the **field trace** (or simply the **trace**, since it is the only trace we will be dealing with).

Definition 6.1. The **trace** of $x \in \mathrm{GF}(2^n)$, denoted $\mathrm{Tr}(x)$, is a mapping from $\mathrm{GF}(2^n)$ into itself, given by

$$\mathrm{Tr}(x) = \sum_{i=0}^{n-1} x^{2^i}.$$

Now that we have it, we'd better figure out what we're allowed to do with it. We will begin by proving that it is not always zero.

Lemma 6.2. *There exists an x in $GF(2^n)$ such that $\mathrm{Tr}(x) \neq 0$.*

Proof. The trace of x can be looked at as a polynomial of degree at most 2^{n-1}. Thus there can be at most 2^{n-1} distinct x such that $\mathrm{Tr}(x) = 0$. But there are 2^n elements in $\mathrm{GF}(2^n)$. Thus there are at least 2^{n-1} elements x in $\mathrm{GF}(2^n)$ such that $\mathrm{Tr}(x) \neq 0$.

□

So we at least know that the trace is not a completely uninteresting function. We will next demonstrate that it is linear.

Proposition 6.3. *For all x, $y \in GF(2^n)$,*

$$\mathrm{Tr}(x + y) = \mathrm{Tr}(x) + \mathrm{Tr}(y).$$

Proof.

$$\begin{aligned}
\mathrm{Tr}(x+y) &= \sum_{i=0}^{n-1} (x+y)^{2^i} \\
&= \sum_{i=0}^{n-1} (x^{2^i} + y^{2^i}) \quad \text{(by the linearity of squaring in } \mathrm{GF}(2^n)) \\
&= \sum_{i=0}^{n-1} (x^{2^i}) + \sum_{i=0}^{n-1} (y^{2^i}) \\
&= \mathrm{Tr}(x) + \mathrm{Tr}(y).
\end{aligned}$$

□

Now, two multiplication-related properties:

Proposition 6.4. *For all $x \in GF(2^n)$,*

$$\mathrm{Tr}(x^2) = \mathrm{Tr}(x).$$

Proof.

$$\begin{aligned}
\mathrm{Tr}(x^2) &= \sum_{i=0}^{n-1} (x^2)^{2^i} \\
&= \sum_{i=0}^{n-1} x^{2^{i+1}} \\
&= \sum_{i=0}^{n-2} x^{2^{i+1}} + x^{2^n} \\
&= \sum_{i=1}^{n-1} x^{2^i} + x \quad (\text{since } x^{2^n} = x \ \forall x \in \mathrm{GF}(2^n)) \\
&= \sum_{i=0}^{n-1} x^{2^i} \\
&= \mathrm{Tr}(x)
\end{aligned}$$

□

Lemma 6.5. *For all $x \in GF(2^n)$,*

$$(\mathrm{Tr}(x))^2 = \mathrm{Tr}(x).$$

Proof.

$$\begin{aligned}
(\mathrm{Tr}(x))^2 &= (\sum_{i=0}^{n-1} x^{2^i})^2 \\
&= \sum_{i=0}^{n-1} (x^{2^i})^2 \\
&= \sum_{i=0}^{n-1} x^{2^{i+1}} \\
&= \mathrm{Tr}(x) \quad \text{by the same logic as the previous proof.}
\end{aligned}$$

□

That's all pretty straightforward and not particularly surprising. However, it gets us the following, less obvious, result:

Proposition 6.6. *For all $x \in GF(2^n)$, $\mathrm{Tr}(x) \in \{0, 1\}$.*

Proof. By Lemma 6.5, we know that

$$\begin{aligned}
&(\mathrm{Tr}(x))^2 = \mathrm{Tr}(x) \quad \forall x \in \mathrm{GF}(2^n) \\
\implies &(\mathrm{Tr}(x))^2 + \mathrm{Tr}(x) = 0 \\
\implies &\mathrm{Tr}(x)(\mathrm{Tr}(x) + 1) = 0 \\
\implies &\mathrm{Tr}(x) = 0 \text{ or } \mathrm{Tr}(x) = 1.
\end{aligned}$$

□

Thus, the trace is a boolean function. Now we will consider what kinds of boolean functions we can make using the trace. We will consider functions of the form $x \mapsto \mathrm{Tr}(\omega x)$ for $\omega \in \mathrm{GF}(2^n)$.

Lemma 6.7. $\mathrm{Tr}(\omega x)$ *is a linear function of* x.

Proof. We need only prove that $\mathrm{Tr}(\omega(x_1 + x_2)) = \mathrm{Tr}(\omega x_1) + \mathrm{Tr}(\omega x_2)$:

$$\begin{aligned}
&\mathrm{Tr}(\omega(x_1 + x_2)) \\
&= \mathrm{Tr}(\omega x_1 + \omega x_2) \\
&= \mathrm{Tr}(\omega x_1) + \mathrm{Tr}(\omega x_2) \quad \text{(by Proposition 6.3).}
\end{aligned}$$

□

Lemma 6.8.

$$\mathrm{Tr}(\omega_1 x) = \mathrm{Tr}(\omega_2 x)\ \forall x \in GF(2^n) \iff \omega_1 = \omega_2$$

Proof. It is clear that if $\omega_1 = \omega_2$, then $\mathrm{Tr}(\omega_1 x) = \mathrm{Tr}(\omega_2 x) \forall x \in \mathrm{GF}(2^n)$. We thus need to prove the reverse implication. For each x in $\mathrm{GF}(2^n)$,

$$\mathrm{Tr}(\omega_1 x) = \mathrm{Tr}(\omega_2 x) \implies \mathrm{Tr}(\omega_1 x) + \mathrm{Tr}(\omega_2 x) = 0 \implies \mathrm{Tr}((\omega_1 + \omega_2)x) = 0.$$

If $\omega_1 \neq \omega_2$, then $x \mapsto (\omega_1 + \omega_2)x$ is an invertible mapping from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2^n)$. That is, it is a permutation. Thus:

$$\begin{aligned}
&\mathrm{Tr}((\omega_1 + \omega_2)x) = 0 \quad \forall x \in \mathrm{GF}(2^n) \\
&\implies \mathrm{Tr}(x) = 0 \quad \forall x \in \mathrm{GF}(2^n)
\end{aligned}$$

But Lemma 6.2 states that there is at least one x such that $\mathrm{Tr}(x) \neq 0$, so this is a contradiction. Thus we can conclude that $\omega_1 = \omega_2$

□

Now we can combine Lemma 6.7 and Lemma 6.8 to arrive at the following result, which is the main reason we're interested in the trace function.

Theorem 6.9. *The set of functions* $\mathrm{Tr}(\omega x)$, $\omega \in GF(2^n)$, *are exactly the set of linear boolean functions on* $GF(2^n)$.

Proof. This result follows directly from the observation that there are exactly 2^n possible linear boolean functions, 2^n distinct values for ω, and Lemmas 6.7 and 6.8

□

This means that we can represent any linear boolean function with a trace function. In particular, for any function F that maps from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2^m)$, each component boolean function of F, and more generally any linear combination of component boolean functions of F, can be represented by $\mathrm{Tr}(\omega F)$ for some ω in $\mathrm{GF}(2^m)$. This will be highly useful in proving the nonlinearity of certain classes of functions.

7. Cryptanalysis

We don now the role of an attacker on a cipher, an eavesdropper on a communication between two entities. Developing a theoretical attack on a cipher is called **cryptanalysis**. Of course, in order to design a cipher, we must first understand what must be defended against, and so the distinction between cryptographer and cryptanalyst is often simply what hat a person happens to be wearing at the time (hence the slang "White Hat" and "Black Hat" to describe the good guys and the bad guys, respectively, in many areas of computer security). Before proceeding with specific attacks we must first define the knowledge we go into an attack on a cipher with, and also the criteria for success in our attack.

There are various assumptions we might make about this situation. The most minimal is that we know nothing- not the algorithm used, the block size, the plaintext of any of the communication, nor the key. We see simply a sequence of bits. However, we don't want the security of a cipher to be based, even in part, on the secrecy of the algorithm used, as this is much more difficult to maintain than the secrecy of a particular key. Thus, it is generally assumed that the algorithm is fully known to the attacker. We, as attacker, are granted much more knowledge than this, however. We are given the power to mount what is called a **known plaintext attack**. This means that we know what some of the plaintext input to the keyed algorithm is. This is not an unusual situation, as much computer-to-computer communication involves header and formatting data (as in email messages) that does not vary from message to message, and can be guessed at by the attacker. It means the attacker has a set of (input, output) pairs to the keyed cipher. It is hard to fully understand what information this grants in general, so this knowledge is extended to a more general **chosen plaintext attack** where it is assumed the attacker can choose inputs to the keyed algorithm and receive the outputs. This is a less common scenario, but it is a superset of the previous one. It gives us as the attacker all data except the key. Security against known plaintext attacks is the strongest requirement for security of a cipher.

Our success is easier to describe. We want the key. Equivalently, we want all the round keys. One way to get this result that is unstoppable and always successful is a brute-force search of the keyspace. That is, test every possible value of the key until we find the one that, when we key the algorithm with it, generates the same output as the keyed algorithm we are attacking (which can be verified with a number of chosen plaintexts that is linear with the length of the key). This attack is on the order of 2^n, where n is the number of bits in the key. It is this universal attack that determines reasonable lower bounds for key lengths (currently around 128 bits). This is not a useful attack, but it is a useful *baseline* for attacks. What we are looking for is an attack that takes less effort than this; for instance, an attack that requires 2^{n-2} (input, output) pairs to calculate all

the round keys. Even if such an attack is not fast enough to recover the key in a practical amount of time, it is still a theoretical weakness in the cipher.

The main goal in designing a cipher, then, is to make all known attacks require more effort than a brute force keyspace search. We will now define several mathematical characteristics the S-box function of a cipher must possess in order for the cipher to have this property.

8. Linear Cryptanalysis

It has been noted that with the exception of the S-box function, the entire encryption algorithm is linear. So if we can approximate the S-box with a linear function, the algorithm is potentially compromised. So what does a linear approximation look like? We want some sort of linear relationship between the input bits and the output bits of the S-box that is true for significantly more than half of all possible inputs. This means that if the S-box is defined by a function $\mathcal{S}$ from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2^m)$, we want vectors $a \in \mathrm{GF}(2^n)$, $b \in \mathrm{GF}(2^m)$, $c \in \mathrm{GF}(2)$, such that $a^{\mathsf{T}}x = b^{\mathsf{T}}\mathcal{S}(x) + c \pmod 2$ for as large a number of input values as possible. Alternately, we can drop the c, and look for either a maximal or minimal number of input values that match the approximation. Another way to express this is to say that the approximation $a^{\mathsf{T}}x = b^{\mathsf{T}}\mathcal{S}(x) \pmod 2$ holds with probability $\frac{1}{2} + \varepsilon$ for some $\varepsilon \in [-\frac{1}{2}, \frac{1}{2}]$, and we're looking for approximations with nonzero ε, the larger in magnitude the better.

Recall that the S-box is only a portion of the round function, and in fact only a portion of the nonlinear transform step. We need to first extend S-box approximations into approximations for the full nonlinear function in the round function. Typically the nonlinear function is applied by breaking the input block into q blocks of equal length, inputting each into a single S-box, and reassembling the outputs of the q S-boxes into a single large block again. The S-boxes may all be the same, or they may not be. So let

$$F(x) = (\mathcal{S}_1(x_1), \mathcal{S}_2(x_2), ..., \mathcal{S}_q(x_q))$$

represent the round's nonlinear function, with the $\mathcal{S}_i$ set representing the q S-box functions. Now, if we have a linear approximation for F of the form

$$a^{\mathsf{T}}x = b^{\mathsf{T}}F(x) \pmod 2$$

that holds with probability $\frac{1}{2} + \varepsilon$, we'd like to know how to relate this back to approximations of individual S-boxes. Partition a and b into q pieces corresponding to the x_is, and rewrite our approximation as

$$\begin{aligned} &(a_1, a_2, ..., a_q)^{\mathsf{T}}(x_1, x_2, ..., x_q) \\ &\qquad = (b_1, b_2, ..., b_q)^{\mathsf{T}}(\mathcal{S}_1(x_1), \mathcal{S}_2(x_2), ..., \mathcal{S}_q(x_q)) \pmod 2 \\ \rightarrow \quad &a_1^{\mathsf{T}}x_1 + a_2^{\mathsf{T}}x_2 + ... + a_q^{\mathsf{T}}x_q \\ &\qquad = b_1^{\mathsf{T}}\mathcal{S}_1(x_1) + b_2^{\mathsf{T}}\mathcal{S}_2(x_2) + ... + b_q^{\mathsf{T}}\mathcal{S}_q(x_q) \pmod 2 \end{aligned}$$

This is the sum of q single S-box linear approximations of the form

$$a_i^{\mathsf{T}} x_i = b_i^{\mathsf{T}} \mathcal{S}_i(x_i) \pmod 2,$$

each having a probability of $(\frac{1}{2}+\varepsilon_i)$. The connection between these probabilities and the probability of the total approximation is given by the following Lemma:

Lemma 8.1. (Piling-Up Lemma, see [11] for origin) *Given q independent approximations of the form $X_i = Y_i \pmod 2$ with probability of occurrence $\frac{1}{2}+\varepsilon_i$, then the approximation $X_1+X_2+...+X_q = Y_1+Y_2+...+Y_q \pmod 2$ holds with probability*

$$\frac{1}{2} + 2^{q-1} \prod_{i=1}^{q} \varepsilon_i.$$

Proof. (by induction) For $q = 2$:

$$\begin{aligned}
&\Pr(X_1 + X_2 = Y_1 + Y_2) \\
&\quad = \Pr(X_1 = Y_1)\Pr(X_2 = Y_2) + \Pr(X_1 \neq Y_1)\Pr(X_2 \neq Y_2) \\
&\quad = \left(\frac{1}{2} + \varepsilon_1\right)\left(\frac{1}{2} + \varepsilon_2\right) + \left(\frac{1}{2} - \varepsilon_1\right)\left(\frac{1}{2} - \varepsilon_2\right) \\
&\quad = \frac{1}{2} + 2\varepsilon_1\varepsilon_2 \pmod 2
\end{aligned}$$

Now, assume true for q:

$$\begin{aligned}
&\Pr(X_1 + X_2 + ... + X_{q+1} = Y_1 + Y_2 + ... + Y_{q+1}) \\
&\quad = \Pr(X_1 + X_2 + ... + X_q = Y_1 + Y_2 + ... + Y_q)\Pr(X_{q+1} = Y_{q+1}) \\
&\qquad + \Pr(X_1 + X_2 + ... + X_r \neq Y_1 + Y_2 + ... + Y_q)\Pr(X_{q+1} \neq Y_{q+1}) \\
&\quad = \left(\frac{1}{2} + 2^{q-1}\prod_{i=1}^{q}\varepsilon_i\right)\left(\frac{1}{2} + \varepsilon_{q+1}\right) + \left(\frac{1}{2} - 2^{q-1}\prod_{i=1}^{q}\varepsilon_i\right)\left(\frac{1}{2} - \varepsilon_{q+1}\right) \\
&\quad = \frac{1}{2} + 2^q \prod_{i=1}^{q+1}\varepsilon_i \pmod 2.
\end{aligned}$$

□

Thus, assuming the inputs to each S-box are independent, which we'll wave our hands at in a minute, this lemma tells us that the probability of the approximation for the nonlinear function is given by $\frac{1}{2} + 2^{q-1}\prod_{i=i}^{q}\varepsilon_i$. An implication of this is that an approximation for the nonlinear function will be better, in general, if the number of S-boxes involved is kept small- that is, if many of the a_i and b_i vectors are equal to zero. In this case the approximations $a_i^{\mathsf{T}} x_i = b_i^{\mathsf{T}} \mathcal{S}_i(x_i) \pmod 2$ becomes $0 = 0 \pmod 2$, which

holds with probability 1, and thus doesn't "weigh down" the probability for the approximation of the nonlinear function.

But how does this get us closer to recovering the key? There are several linear functions composed with the nonlinear function to make up the round function, including a key addition, which is generally carried out as a simple bitwise addition of the round key to the block of text being operated on. So, either the input to the nonlinear function will be a linear combination of the input to the round function and the round key, or the output of the round function will be a linear combination of the output of the nonlinear function and the round key. So, for example, a round function $R(x, k)$ may be written as:

$$R(x, k) = L_2(F(L_1(x) + k)) \tag{4}$$

where x is the input to the round, k is the round key for that round, L_1 and L_2 are linear functions, and F is the nonlinear function made up by applying the q S-box functions in parallel. Of course the linear approximations we're using, when applied to linear functions, hold true with probability 1, so the approximation can be extended to become an approximation for the entire round, involving the inputs to the round, x and k, and the output of the round, $y = R(x, k)$. With some rearranging of the sides of the equation, the approximations will take the form $d^{\mathsf{T}}k = a^{\mathsf{T}}x + b^{\mathsf{T}}y \pmod 2$, holding, again, with some probability $(\frac{1}{2}+\varepsilon)$. The larger the magnitude of ε, the more useful an approximation it is in recovering data about the bits of the round key selected by d, essentially recovering one bit of information about the round key. With more such approximations it is possible to recover more data. In [11] it is shown that ε^{-2} (input, output) pairs are sufficient to be roughly 99% confident that the key information suggested by the approximation is correct.

This, of course, is simply breaking one round- we must break many (usually in the low double-digits range). This means worrying about intermediate values for the block of input. What we are after are **linear trails**.

Definition 8.2. An **r-round linear trail** consists of r approximations of the form $d_i^{\mathsf{T}}k_i = a_i^{\mathsf{T}}x + b_i^{\mathsf{T}}y \pmod 2$, with the requirement that the b_i selection vector on the output of one approximation equal the a_{i+i} selection vector on the input of the next approximation.

By adding these r approximations mod 2 all of the intermediate terms cancel, and we are left with a single large approximation involving only inputs to the first round, outputs of the rth round, and the round keys. If we assume the round keys are statistically independent of one another (which in practice is almost true- that being the goal of the key schedule- and thus makes for at least a very good approximation), then they make the inputs to each round and each S-box independent as well. Thus we can apply the Piling-up Lemma to this final approximation to determine its probability, in terms, ultimately, of the probabilities of each S-box approximation.

This, then, is **linear cryptanalysis**: find a sufficient number of linear trails to recover most of the key, then generate enough (plaintext, ciphertext) pairs to determine those key bits with a reasonable degree of certainty. Finally, use a small amount of brute-force key searching to recover the final few bits that the approximations didn't yield. Result: full key recovery, and the breaking of the cipher, assuming of course that all this work was in fact *less* work than doing a full keyspace search to begin with.

So, what does the preceding tell us about making a cipher resistant to this attack? We want the ε associated with any linear trail to be small enough that ε^{-2} is larger than the total number of possible keys, so that generating a sufficient number of (plaintext, ciphertext) pairs takes longer than a brute-force search of the keyspace. There are two ways to do this. One is to choose the linear functions involved in the round function to make sure that as many S-boxes are involved with any trail as possible. This would make it difficult for an attacker to find a trail through only a few S-boxes, which would be more likely to have a large ε. We are focusing on the S-box design, however, so we'll pass over that. The other defense is to choose S-box functions so that the ε associated with each possible linear approximation of the S-box is bounded by as small a value as possible. This forces the ε for any linear trail to be bounded in a related way, through the Piling-up Lemma.

Let's build some formalism to deal with this goal of a tiny ε. The following definitions and results are largely from [13].

Definition 8.3. For boolean function $f : \mathrm{GF}(2^n) \to \mathrm{GF}(2)$ the **nonlinearity** of f is:

$$\mathcal{N}(f) = \min_{\substack{a \in \mathrm{GF}(2^n) \\ c \in \mathrm{GF}(2)}} \#\{x \in \mathrm{GF}(2^n) : f(x) = a^{\mathsf{T}} x + c\}$$

where # represents cardinality.

This should look familiar: it is $d(f, \mathcal{A}_n)$, the Hamming distance of f from the affine functions, our previous definition of nonlinearity for boolean functions. Now we will extend this from boolean functions to the general case.

Definition 8.4. For function $F : \mathrm{GF}(2^n) \to \mathrm{GF}(2^m)$ the **nonlinearity** of F is:

$$\begin{aligned} \mathcal{N}(F) &= \min_{\substack{b \in \mathrm{GF}(2^m) \\ b \neq 0}} \mathcal{N}(b^{\mathsf{T}} F) \\ &= \min_{\substack{a \in \mathrm{GF}(2^n),\ b \in \mathrm{GF}(2^m) \\ c \in \mathrm{GF}(2),\ b \neq 0}} \#\{x \in \mathrm{GF}(2^n) : b^{\mathsf{T}} F(x) = a^{\mathsf{T}} x + c\} \end{aligned}$$

where # represents cardinality.

Another way to state this is to say that the nonlinearity of a function

$$F(x) = (f_1(x), f_2(x), ..., f_m(x))$$

is the minimum of the nonlinearities of all linear combinations of its component functions. From Theorem 5.7, we can take the nonlinearity of the perfectly nonlinear boolean functions, $2^{n-1} - 2^{\frac{n}{2}-1}$, as an upper bound on $\mathcal{N}(F)$ in general.

Proposition 8.5.

$$\mathcal{N}(F) = \min_{\substack{a \in GF(2^n),\ b \in GF(2^m) \\ c \in GF(2),\ a \neq 0 \text{ or } b \neq 0}} \#\{x \in GF(2^n) : b^{\mathsf{T}} f(x) = a^{\mathsf{T}} x + c\}$$

Proof. If $b = 0$, then

$$\begin{aligned} &\min_{\substack{a \in \mathrm{GF}(2^n), b \in \mathrm{GF}(2^m) \\ c \in \mathrm{GF}(2), a \neq 0 \text{ or } b \neq 0}} \#\{x \in \mathrm{GF}(2^n) : b^{\mathsf{T}} f(x) = a^{\mathsf{T}} x + c\} \\ =&\min_{\substack{a \in \mathrm{GF}(2^n), c \in \mathrm{GF}(2) \\ a \neq 0}} \#\{x \in \mathrm{GF}(2^n) : 0 = a^{\mathsf{T}} x + c\} \\ =&\frac{1}{2} 2^n = 2^{n-1} \geq \mathcal{N}(F). \end{aligned}$$

Thus the inclusion of these extra elements in the set will not change the set's minimum.

□

Using this slight generalization, we can say something about how $\mathcal{N}(f)$ behaves under the taking of inverses:

Theorem 8.6. *Let $F : GF(2^n) \to GF(2^n)$ be a permutation. Then*

$$\mathcal{N}(F^{-1}) = \mathcal{N}(F).$$

Proof.

$$\begin{aligned} &N(F^{-1}) \\ &\quad = \min_{\substack{a,b \in \mathrm{GF}(2^n),\ c \in \mathrm{GF}(2) \\ a \neq 0 \text{ or } b \neq 0}} \#\{y \in \mathrm{GF}(2^n) : b^{\mathsf{T}} F^{-1}(y) = a^{\mathsf{T}} y + c\} \\ &\quad = \min_{\substack{a,b \in \mathrm{GF}(2^n),\ c \in \mathrm{GF}(2) \\ a \neq 0 \text{ or } b \neq 0}} \#\{x \in \mathrm{GF}(2^n) : b^{\mathsf{T}} x = a^{\mathsf{T}} F(x) + c\} \\ &\quad = \mathcal{N}(F) \end{aligned}$$

□

Using these results, we can say that for any linear approximation of F,

$$|\varepsilon| \leq \frac{2^n - \mathcal{N}(F)}{2^n} - \frac{1}{2} = \frac{1}{2} - 2^{-n}\mathcal{N}(F).$$

By extension, in choosing an S-box function $\mathcal{S}$, we want something with as large a value for $\mathcal{N}(\mathcal{S})$ as possible. As the lowest upper bound we have so far is $2^{n-1} - 2^{\frac{n}{2}-1}$, we will try to find functions whose nonlinearity is close to that.

So, given all this, we can formally state one of the goals in designing secure S-boxes:

Goal A. A suitable S-box function $\mathcal{S}$: $\mathrm{GF}(2^n) \to \mathrm{GF}(2^m)$ should have the property that $\mathcal{N}(\mathcal{S})$ is maximal or near-maximal. Specifically, $\mathcal{N}(\mathcal{S})$ should be close to the upper bound $2^{n-1} - 2^{\frac{n}{2}-1}$.

9. Differential Cryptanalysis

In general, any time we can find a way to recognize patterns in the non-linear S-box, and propagate information about those patterns out of the algorithm, we have a potential vulnerability in the cipher. One such pattern is how differences in input to the S-box function relate to differences in its output. The natural way to represent "difference" is, again, hamming distance. In 1990 Eli Biham and Adi Shamir proposed (in [1], and in 1993 refined in [2]) using the statistical relationship between these differences to recover round keys in the DES cipher.

We will begin by considering a single S-box function

$$\mathcal{S}(x) : \mathrm{GF}(2^n) \to \mathrm{GF}(2^m).$$

Given two inputs to $\mathcal{S}$, x_1 and x_2, and their related outputs $y_1 = \mathcal{S}(x_1)$ and $y_2 = \mathcal{S}(x_2)$, we can calculate

$$x' = x_1 + x_2$$
$$y' = y_1 + y_2.$$

The relevant question then, is: for a given input difference x', are all output differences y' possible, or equally likely? The answer is no, and in the case of the vast majority of possible S-box functions, not even close. We can generate a table relating input and output differences in the following way:

Fix x'. $\forall x \in \mathrm{GF}(2^n)$:
  Find $y' = \mathcal{S}(x + x') + \mathcal{S}(x)$
  Increment table position (x', y')
Repeat for all x'.

In the end we will have a table, called a **pairs xor distribution table**, with 2^n rows, corresponding to the input differences, and 2^m columns, corresponding to the output differences. All entries in the table will be even, since

$$\mathcal{S}(x + x') + \mathcal{S}(x) = \mathcal{S}((x + x') + x') + \mathcal{S}(x + x'). \tag{5}$$

Also, the entries in each row (a fixed x') sum to 2^n. Many entries will be zero, meaning that for a given input difference, certain output differences never occur. If table entry $(x', y') = q \neq 0$, we say that input difference x' suggests output difference y' with probability $\frac{q}{2^n}$. If table entry $(x', y') = 0$,

we say that input difference x' does not suggest output difference y'. On the row corresponding to an input difference $x' = 0$, the output difference $y' = 0$ will have value 2^n, and all other output differences will have a value of 0, since identical inputs to the S-box must lead to identical outputs.

Recall that the nonlinear substitution function F in the round function of a cipher is typically made up of several S-boxes operating on pieces of the input in parallel. The probability of an input difference to F suggesting a particular output difference is simply the product of the probabilities that each S-box's input difference suggests its output difference.

So now we have a way of relating input differences to output differences in the only nonlinear component of the cipher. We will now extend it to a full round. Recall the round function R given in (4), and consider round inputs $x_1, x_2 \in \mathrm{GF}(2^n)$, with $x' = x_1 + x_2$, and round outputs $y_1 = R(x_1, k)$ and $y_2 = R(x_2, k)$, with $y' = y_1 + y_2$. Then

$$L_1(x_1) + L_1(x_2) = L_1(x_1 + x_2) = L_1(x').$$

When the key is added to both intermediate values, it is cancelled out when adding them together mod 2, so the difference is invariant under key addition. Thus, by knowing the difference in round inputs, we can determine the difference in S-box inputs, regardless of key. Now,

$$\begin{aligned}
&F(L_1(x_1) + k) + F(L_1(x_2) + k) \\
=&L_2^{-1}(L_2(F(L_1(x_1) + k) + F(L_1(x_2) + k))) \\
=&L_2^{-1}(L_2(F(L_1(x_1) + k)) + L_2(F(L_1(x_2) + k))) \\
=&L_2^{-1}(y_1 + y_2) \\
=&L_2^{-1}(y').
\end{aligned}$$

Thus we can also determine the difference in nonlinear function outputs given the difference in round outputs. What all this means is that we can extend our description of F to a description of the full round function, with no further reduction in probabilities.

We now need to leverage these mechanics to recover the round key. So, for a given (x', y') round input/output difference pair, we can determine an (x'_F, y'_F) nonlinear function difference pair. Let's go to work:

For a keyed round function R_k, round input x, and input difference x', calculate:

$$\begin{aligned}
y &= R_k(x) \\
y' &= R_k(x) + R_k(x + x') \\
x'_F &= L_1(x') \\
y'_F &= L_2^{-1}(y').
\end{aligned}$$

Now, find the set $X_F = \{x_F : F(x_F) + F(x_F + x'_F) = y'_F\}$. This is the set of all the possible inputs to the function F that satisfy the constraint that the input difference and output difference agree with x'_F and y'_F determined by

our choice of x and x'. The cardinality of X_F will be the number at position (x', y') on the pairs xor distribution table of F. Each of these possible values for x_F suggests a possible value for the round key in the following way: since, for the actual value of x_F, $x_F = k + L_1(x)$,

$$k = x_F + L_1(x). \tag{6}$$

So we have $|X_F|$ possible values for k. Now we repeat the process by varying our choice of x, or x', or both, as desired. The actual round key will be the one suggested by *all* trials. Other suggested key values will be distributed in a generally random fashion, though for a given x', the actual key value k and $k + L_1(x')$ will both be suggested. Thus multiple choices of x' are required to uniquely identify the round key.

So, we can break one round without too much difficulty, and also without invoking the probabilities associated with each (x', y') pair. Now we must extend this attack to an arbitrary number of rounds in order to break a full cipher. We do this by means of **differential characteristics**.

Definition 9.1. An r-**round differential characteristic** is a set of r differences

$$\{x'(1), x'(2), ..., x'(r)\}$$

such that for $i = 1, 2, ..., (r-1)$, the difference $x'(i)$ suggests the difference $x'(i+1)$ in round r with nonzero probability.

What we have, then, is the equivalent of a linear trail in linear cryptanalysis. Given a pair of cipher inputs x_1 and x_2 such that $x_1 + x_2 = x'(1)$, a differential characteristic is a possible set of differences that the intermediate values generated by x_1 and x_2 will have, so $R(x_1) + R(x_2)$ may equal $x'(2)$, and so on through the cipher, until the final output difference at the end of r rounds may be $x'(r)$. That is a lot of "may"s. Assuming that the round keys are uniformly random (which, again, is close to, but not exactly, true), the probability of a characteristic holding true for a given $x_1, (x_1 + x'(1))$ input pair is the product of the probabilities of it holding true at each round. That is, if p_i is the probability that $x'(i)$ suggests $x'(i+1)$ in round i, then the probability of a characteristic holding for a given input pair is:

$$\prod_{i=1}^{r-1} p_i.$$

Now, we have determined that, to recover a round key, we need a set of pairs of inputs to that round, a set of input differences, and a set of output differences. If we have an input pair for which a characteristic holds, then we have all the input differences and output differences for each round. We also have the input value to the first round. Thus, if the set is large enough, we can recover the first round's key. Using that key, we can determine the input to the second round, and recover its key in the same way, and so forth through the cipher.

However, how do we know if a characteristic holds? Once we have constructed an r-round characteristic which holds with probability p, we can generate a large set of input pairs with the correct difference, and encrypt all of them. For each pair of outputs, we can check their difference against the final difference in the characteristic. If they differ, then that input pair is not a "right pair" for that characteristic, and can be discarded. Almost all pairs will fall into this category. Of the remaining pairs, all have the correct initial difference value x' and final difference value y'. We cannot know, however, if all intermediate difference values are correct. Thus when we try to recover the key from the first round, there will not necessarily be a key value that is suggested by all the difference pairs. However, the "wrong pairs" will suggest key values that are reasonably randomly distributed, just as the wrong key values suggested by the "right pairs" are. Thus the correct round key is likely to be the value suggested *most often* by all the remaining pairs.

So, given enough input pairs and corresponding output pairs, we can recover the round keys. But how much is enough? And, more importantly, is it more or less than would be required by a brute force search of the keyspace? The number of "right pairs" needed will be some constant value determined primarily by the input size, as well as by the probability that any involved 1-round characteristic will hold. The number of "right pairs" found out of a set of input pairs will be determined by the probability of the r-round characteristic. Thus the dominating factor in determining whether or not differential cryptanalysis can be successful against a cipher is the maximum probability of any differential characteristic in that cipher. Two factors determine this maximum. One is the minimum number of S-boxes active in any characteristic. An S-box is **active** if the input to it is different for the two inputs to the round. If both inputs to the round only differ in the second half of their bits after being fed through the linear function L_1, then the S-boxes acting on the first half of their bits are seeing an input difference of 0, which will lead to an output difference of 0, with probability 1. Thus when choosing a characteristic it is beneficial to pick differences that only affect a limited number of bits (the downside being that an inactive S-box can reveal nothing about the key bits associated with it). Thus it is the job of the linear functions L_1 and L_2 to diffuse differences and force as many S-boxes to be active as possible, thus reducing the probability of any characteristic. The second determining factor is the maximum probability of any input difference to an S-box leading to any output difference.

It is this second requirement that we will concern ourselves with, as it is the one the choice of S-box has control over. We will formalize the requirement in the following way:

Definition 9.2. A function $F : \mathrm{GF}(2^n) \to \mathrm{GF}(2^m)$ is called **differentially δ-uniform** if, for all $\alpha \in \mathrm{GF}(2^n), \beta \in \mathrm{GF}(2^m), \alpha \neq 0$,

$$\#\{x \in \mathrm{GF}(2^n) : F(x+\alpha) + F(x) = \beta\} \leq \delta$$

where # represents cardinality.

This means that no entry in the pairs xor distribution table can be larger than δ. If a cipher is differentially δ-uniform, and in any characteristic at least s S-boxes must be active, then the maximum probability for any characteristic must be less than or equal to $\left(\frac{\delta}{2^n}\right)^s$. The number of plaintext pairs needed to be encrypted to launch a differential attack will be $c(\frac{2^n}{\delta})^s$ for some constant c. We can now use this definition to define our second goal.

Goal B. A suitable S-box function $\mathcal{S}$: $\mathrm{GF}(2^n) \to \mathrm{GF}(2^m)$ should be differentially δ-uniform for as small a δ as possible.

10. Power Polynomials

Now that we have a pair of goals in mind, we can begin looking for potential S-box functions. While the squaring function $x \mapsto x^2$ on $\mathrm{GF}(2^n)$ is linear, it was noted (see, for instance, [16]) that certain other power functions, such as $x \mapsto x^3$, appeared to be highly suitable candidates for S-box functions. Through the course of further study (see [13], [14], [15]), various properties of functions of the form $x \mapsto x^{2^k+1}$, on $\mathrm{GF}(2^n)$, when a permutation, were proven. These properties make functions of this form satisfy our two goals to a near-maximal extent. We will now run through the proof of this, the main structure of which is taken from [14].

Proposition 10.1. *Let $F(x) = x^{2^k+1}$ be a power polynomial acting on elements of $GF(2^n)$ and let $s = \gcd(k, n)$. Then F is differentially 2^s-uniform.*

Proof. We aim to show that

$$\max_{\substack{\alpha,\beta\in\mathrm{GF}(2^n)\\ \alpha\neq 0}} \{x : F(x+\alpha) + F(x) = \beta\} \leq 2^s.$$

So, let $\alpha, \beta \in \mathrm{GF}(2^n)$, $\alpha \neq 0$ be arbitrary. The equation

$$(x+\alpha)^{2^k+1} + x^{2^k+1} = \beta$$

has either zero or at least two solutions, since solutions come in pairs $(x, x+\alpha)$. Let x_1, x_2 be two distinct solutions. Thus:

$$\begin{aligned}
&(x_1+\alpha)^{2^k+1} + x_1^{2^k+1} + (x_2+\alpha)^{2^k+1} + x_2^{2^k+1} = \beta + \beta \\
&\implies (x_1+\alpha)(x_1^{2^k} + \alpha^{2^k}) + x_1^{2^k+1} + (x_2+\alpha)(x_2^{2^k} + \alpha^{2^k}) + x_2^{2^k+1} = 0 \\
&\implies \alpha x_1^{2^k} + x_1\alpha^{2^k} + \alpha x_2^{2^k} + x_2\alpha^{2^k} = 0 \\
&\implies \alpha(x_1+x_2)^{2^k} + \alpha^{2^k}(x_1+x_2) = 0 \\
&\implies (x_1+x_2)^{2^k-1} + \alpha^{2^k-1} = 0 \quad (\text{since } \alpha(x_1+x_2) \neq 0) \\
&\implies (x_1+x_2)^{2^k-1} = \alpha^{2^k-1}.
\end{aligned}$$

This means that $(x_1 + x_2) = \alpha\gamma$, where $\gamma^{2^k-1} = 1$. We must determine how many possible values there are for γ. Since $\mathrm{GF}(2^n)\backslash\{0\}$ forms a group under multiplication of order 2^n-1, the multiplicative order of γ must divide $2^n - 1$. In order for $\gamma^{2^k-1} = 1$ to hold, the order of γ must divide $2^k - 1$. Thus the order of γ must divide $\gcd(2^n - 1, 2^k - 1) = 2^{\gcd(n,k)} - 1 = 2^s - 1$ (by Knuth's GCD lemma- see, for instance, [8]). The set of all such γ, with 0 added, forms a subfield of $\mathrm{GF}(2^n)$ of order 2^s. Thus there are $2^s - 1$ possible nonzero values for γ ($\gamma = 0$ corresponding to $x_1 = x_2$, which we are discounting). Thus given one solution, there are exactly $2^s - 1$ other solutions, for a total of 2^s. Thus F is differentially 2^s-uniform.

□

Proposition 10.2. *Let $F(x) = x^{2^k+1}$ be a power polynomial acting on elements of GF(2^n) and let $s = \gcd(k, n)$. If F is a permutation, then*

$$\mathcal{N}(F) = 2^{n-1} - 2^{\frac{n+s}{2}-1}.$$

Proof. By Theorem 6.9, any nontrivial linear combination of the component boolean functions of F can be represented by

$$f_\omega(x) = \mathrm{Tr}(\omega F(x))$$

for some $\omega \in \mathrm{GF}(2^n)$, $\omega \neq 0$. Denote by $\hat{\mathrm{F}}_\omega(t)$ the Walsh transform of $f_\omega(x)$. Since, by (3),

$$\mathcal{N}(F) = 2^{n-1} - \frac{1}{2}\left(\max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} |\hat{\mathrm{F}}_\omega(t)|\right)$$

it will suffice to prove that

$$\max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} \left((\hat{\mathrm{F}}_\omega(t))^2\right) = 2^{n+s}.$$

Let $\omega \in \mathrm{GF}(2^n)$, $\omega \neq 0$ be arbitrary. Before we begin manipulating $(\hat{\mathrm{F}}_\omega(t))^2$, we must develop a tool to assist us.

Let $y \in \mathrm{GF}(2^n)$, $y \neq 0$ be arbitrary, and denote by E_y the range of the mapping

$$x \to F(x+y) + F(x) + F(y) = x^{2^k} y + y^{2^k} x. \tag{7}$$

Note that this mapping is linear on x. As in Proposition 10.1, the dimension of the kernel of this mapping is s, so the dimension of E_y is $n - s$.

Now, consider $\mathrm{Tr}(\omega\beta)$ as a linear function of $\beta \in E_y$. One of two things must be true, given this linearity: either:

$$\sum_{\beta \in E_y} (-1)^{\mathrm{Tr}(\omega\beta)} = 0 \tag{8}$$

or:

$$\mathrm{Tr}(\omega\beta) = 0 \quad \forall \beta \in E_y \tag{9}$$

depending on whether $\mathrm{Tr}(\omega\beta)$ is the zero function or not. Let Y be the linear subspace of $\mathrm{GF}(2^n)$ containing all y such that $\mathrm{Tr}(\omega\beta)$ is the zero function. By manipulating $\mathrm{Tr}(\omega\beta)$ further, we see that

$$\begin{aligned}
&\mathrm{Tr}(\omega\beta) \\
&= \mathrm{Tr}(\omega(F(x+y) + F(x) + F(y))) \\
&= \mathrm{Tr}(\omega F(x+y)) + \mathrm{Tr}(\omega F(x)) + \mathrm{Tr}(\omega F(y)) \\
&= f_\omega(x+y) + f_\omega(x) + f_\omega(y).
\end{aligned}$$

So for all y in Y,

$$f_\omega(x+y) = f_\omega(x) + f_\omega(y) \ \forall x \in \mathrm{GF}(2^n).$$

Thus f_ω is linear on Y. But what is the cardinality of Y?

$$\begin{aligned}
\forall y \in Y,\ 0 &= \mathrm{Tr}(\omega\beta) \\
&= \mathrm{Tr}(\omega(x^{2^k} y + y^{2^k} x)) \\
&= \mathrm{Tr}(\omega x^{2^k} y) + \mathrm{Tr}(\omega y^{2^k} x).
\end{aligned}$$

Thus

$$\mathrm{Tr}(\omega x^{2^k} y) = \mathrm{Tr}(\omega y^{2^k} x) \ \forall y \in Y \tag{10}$$

However, by Proposition 6.4, we know that

$$\mathrm{Tr}(\omega y^{2^k} x) = \mathrm{Tr}((\omega y^{2^k} x)^{2^k}) = \mathrm{Tr}(\omega^{2^k} y^{2^{2k}} x^{2^k}). \tag{11}$$

Combining (10) and (11), we get:

$$\mathrm{Tr}(\omega y x^{2^k}) = \mathrm{Tr}(\omega^{2^k} y^{2^{2k}} x^{2^k}) \quad \forall x \in \mathrm{GF}(2^n).$$

Thus, by Lemma 6.8,

$$\begin{aligned}
&\omega y = \omega^{2^k} y^{2^{2k}} \\
\implies &1 = \omega^{2^k - 1} y^{2^{2k} - 1} \\
\implies &1 = (\omega F(y))^{2^k - 1}.
\end{aligned}$$

By the same logic as Proposition 10.1, there are 2^s possible values for $\omega F(y)$. Since F is a permutation, there are thus 2^s possible values for y. One is zero, so we end up with $2^s - 1$ elements in Y.

Now we have the tools needed to manipulate $(\hat{\mathrm{F}}_\omega(t))^2$:

$$
\begin{aligned}
(\hat{\mathrm{F}}_\omega(t))^2 &= \left(\sum_{x\in \mathrm{GF}(2^n)} (-1)^{f_\omega(x)+x^{\mathsf{T}}t}\right)\left(\sum_{y\in \mathrm{GF}(2^n)} (-1)^{f_\omega(y)+y^{\mathsf{T}}t}\right)\\
&= \sum_{x\in \mathrm{GF}(2^n)} \left((-1)^{f_\omega(x)+x^{\mathsf{T}}t}\left(\sum_{y\in \mathrm{GF}(2^n)} (-1)^{f_\omega(x+y)+(x+y)^{\mathsf{T}}t}\right)\right)\\
&= \sum_{y\in \mathrm{GF}(2^n)} \left((-1)^{y^{\mathsf{T}}t}\sum_{x\in \mathrm{GF}(2^n)} (-1)^{f_\omega(x+y)+f_\omega(x)}\right)\\
&= 1\sum_{x\in \mathrm{GF}(2^n)} (1) + \sum_{\substack{y\in \mathrm{GF}(2^n)\\ y\neq 0}} \left((-1)^{y^{\mathsf{T}}t}\sum_{x\in \mathrm{GF}(2^n)} (-1)^{f_\omega(x+y)+f_\omega(x)}\right)\\
&= 2^n + \sum_{\substack{y\in \mathrm{GF}(2^n)\\ y\neq 0}} \left((-1)^{y^{\mathsf{T}}t}\sum_{x\in \mathrm{GF}(2^n)} (-1)^{\mathrm{Tr}(\omega\beta)+f_\omega(y)}\right)\\
&= 2^n + \sum_{\substack{y\in Y\\ y\neq 0}} \left((-1)^{y^{\mathsf{T}}t+f_\omega(y)}\sum_{x\in \mathrm{GF}(2^n)} (-1)^{\mathrm{Tr}(\omega\beta)}\right)\\
&\qquad + \sum_{\substack{y\notin Y\\ y\neq 0}} \left((-1)^{y^{\mathsf{T}}t+f_\omega(y)}\sum_{x\in \mathrm{GF}(2^n)} (-1)^{\mathrm{Tr}(\omega\beta)}\right)\\
&= 2^n + \sum_{\substack{y\in Y\\ y\neq 0}} \left((-1)^{y^{\mathsf{T}}t+f_\omega(y)}\sum_{x\in \mathrm{GF}(2^n)} (1)\right) + \sum_{\substack{y\notin Y\\ y\neq 0}} (-1)^{y^{\mathsf{T}}t+f_\omega(y)}(0)\\
&= 2^n + 2^n \sum_{\substack{y\in Y\\ y\neq 0}} (-1)^{y^{\mathsf{T}}t+f_\omega(y)}.
\end{aligned}
$$

Since f_ω is linear on Y, there exists a t in $\mathrm{GF}(2^n)$ such that $y^{\mathsf{T}}t = f_\omega(y)$. Thus

$$\max_{t\in \mathrm{GF}(2^n)} (\hat{\mathrm{F}}_\omega(t))^2 = 2^n + 2^n \sum_{\substack{y\in Y\\ y\neq 0}} (-1)^0 = 2^n + 2^n(2^s - 1) = 2^{n+s}.$$

Therefore:

$$\max_{t\in \mathrm{GF}(2^n)} \left(\hat{\mathrm{F}}_\omega(t)\right) = 2^{\frac{n+s}{2}}$$

Which gives us what we want, specifically:

$$\mathcal{N}(F) = 2^{n-1} - 2^{\frac{n+s}{2}-1}.$$

□

So, on $\mathrm{GF}(2^n)$, $F(x) = x^{2^k+1}$, when it is a permutation, is a very solid choice for an S-box function, being near-optimal in both nonlinearity and differential uniformity. In fact, the cipher C^* is built on exactly these functions. Also, given Theorem 8.6, the inverses of these functions are equally suitable, while having several added benefits that are outside the scope of this paper.

11. The Inversion Function

In the process of investigating power polynomials and their inverses, another function was found with the desired properties: the **inversion function**. Specifically:

$$F(x) = \begin{cases} x^{-1} & x \neq 0 \\ 0 & x = 0 \end{cases} \tag{12}$$

By plugging the undefined hole in the inversion function, we get a permutation on $\mathrm{GF}(2^n)$. Note that in the field $\mathrm{GF}(2^n)$, $x^{-1} = x^{2^n-2}$ for nonzero x, so this is still more or less a power polynomial. Now to demonstrate the suitability of this function. The main structure of the following proofs again comes from [14].

Proposition 11.1. *Let $F(x)$ be the inversion function as defined in* (12)*. Then F is differentially 4-uniform.*

Proof. As in 10.1, we are interested in the maximum number of solutions to the equation

$$F(x+\alpha) + F(x) = \beta. \tag{13}$$

For arbitrary α and β in $\mathrm{GF}(2^n)$, $\alpha \neq 0$. If $x = 0$ or $x = \alpha$ is a solution to (13), then both are solutions, and $\beta = \alpha^{-1}$. In that case, (13) is equivalent to

$$\begin{aligned} &(x+\alpha)^{-1} + x^{-1} + \alpha^{-1} = 0 \\ \implies &(x)(x+\alpha)(\alpha)((x+\alpha)^{-1} + x^{-1} + \alpha^{-1}) = 0 \\ \implies &(x)(\alpha) + (x+\alpha)(\alpha) + (x)(x+\alpha) = 0 \\ \implies &x^2 + \alpha x + \alpha^2 = 0. \end{aligned}$$

By squaring this and applying the substitution $x^2 = \alpha^2 + \alpha x$ we arrive at

$$x(x^3 + \alpha^3) = 0. \tag{14}$$

If $\gcd(3, 2^n - 1) = 1$, then there are no additional solutions to (13). However, if 3 divides $2^n - 1$, then (14) can be factored further, into

$$x(x+\alpha)(x+\alpha^{1+\frac{1}{3}(2^n-1)})(x+\alpha^{1+\frac{2}{3}(2^n-1)}). \tag{15}$$

for a total of four solutions to (13).

Now, if $x = 0$ and $x = \alpha$ are not solutions to (13), then it is equivalent to

$$\beta x^2 + \alpha\beta x + \alpha = 0 \tag{16}$$

which has at most two solutions in $\mathrm{GF}(2^n)$. Since there are at most four solutions in any circumstance, F is differentially 4-uniform.

□

Proposition 11.2. *Let $F(x)$ be the inversion function as defined in* (12)*. Then*

$$\mathcal{N}(F) \geq 2^{n-1} - 2^{\frac{n}{2}}.$$

Proof. As in 10.2, it will suffice to show that

$$\max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} |\hat{\mathrm{F}}_\omega(t)| \leq 2^{\frac{n}{2}+1}.$$

Consider the substitution $\alpha = tx, c = \omega t$, and note that, since we're interested in the max over all t, we can replace $t^{\mathsf{T}}x$ with $\mathrm{Tr}(tx)$. The Walsh transform then becomes:

$$\begin{aligned}
\max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} \left|\hat{\mathrm{F}}_\omega(t)\right| &= \max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} \left|\sum_{x\in\mathrm{GF}(2^n)} (-1)^{f_\omega(x)+x^{\mathsf{T}}t}\right| \\
&= \max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} \left|\sum_{x\in\mathrm{GF}(2^n)} (-1)^{\mathrm{Tr}(\omega x^{-1})+\mathrm{Tr}(tx)}\right| \\
&= \max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} \left|\sum_{\alpha\in\mathrm{GF}(2^n)} (-1)^{\mathrm{Tr}(\omega t\alpha^{-1})+\mathrm{Tr}(\alpha)}\right| \\
&= \max_{\substack{c\in\mathrm{GF}(2^n)\\ c\neq 0}} \left|\sum_{\alpha\in\mathrm{GF}(2^n)} (-1)^{\mathrm{Tr}(c\alpha^{-1})+\mathrm{Tr}(\alpha)}\right| \\
&= \max_{\substack{c\in\mathrm{GF}(2^n)\\ c\neq 0}} \left|\sum_{\alpha\in\mathrm{GF}(2^n)} (-1)^{\mathrm{Tr}(c\alpha^{-1}+\alpha)}\right|.
\end{aligned}$$

The Walsh transform in this form is seen to be a Kloosterman sum. See [4] for a treatment of these. It is proven there that a reasonably tight upper bound for the absolute value of such a sum is $2\sqrt{2^n}$. Thus

$$\max_{\substack{\omega,t\in\mathrm{GF}(2^n)\\ \omega\neq 0}} |\hat{\mathrm{F}}_\omega(t)| \leq 2^{\frac{n}{2}+1}$$

which is exactly what we want.

□

So, the inversion function (12) is another eminently suitable function to use as an S-box function. It is slightly less nonlinear than the power polynomials discussed in section 10, but it is not restricted to finite fields of odd dimension, as the power polynomials are (in order to be a permutation). This is convenient to the designer of a cipher in that it is natural to define block sizes as a certain number of bytes (1 byte = 8 bits). The cipher Rijndael, which we will discuss momentarily, uses the inversion function.

12. Rijndael

When it was first decided, in the early 1970s, that there should be a "standard" cipher for the US government (and thus a de facto industry standard, since the government is in general a very large purchaser of cryptographic products), there was very little public research going on in cryptography. The National Security Agency (NSA) did not want to produce a cipher for public use, as they feared that this would expose too much of their internal methodology to external scrutiny. Thus the call went out for candidate submissions. IBM was one of the few places where significant work was being done in this area, and they developed an algorithm based on a previously existing cipher called Lucifer, and submitted it for consideration. Upon selection by the National Bureau of Standards (now called the National Institute of Standards and Technology, or NIST), it was dubbed the Data Encryption Standard, or DES. Some tweaks made to the S-boxes of DES at the last minute, allegedly at the behest of the NSA, led to over a decade of paranoia on the part of the newly developing public cryptography field, which invested a great deal of effort in hunting for trapdoors in the algorithm that might reveal key bits to those who knew where to look for them. As it turned out, the tweaks had actually made DES very resistant to differential cryptanalysis, which didn't come along publicly until 1990. This cemented the mystique, still in force today, of the NSA as being 10 to 15 years ahead of the public sector in cryptanalysis.

By the early 1990s DES was demonstrated to be slightly vulnerable to both differential and linear cryptanalysis. More importantly than that, however, was its key length, which could be set to either 40 or 56 bits, neither of which was, by then, long enough. In 1993 a machine was proposed that would cost one million dollars and be able to perform a full brute force keyspace search in 7 hours. With the advance of technology it only became easier and cheaper with time. Various attempts to extend the key length were not satisfactory. DES was no longer sufficient. So in 1997 the call once again went out for candidates for a new standard, to be called the Advanced Encryption Standard, or AES. NIST released a set of design requirements, and in return received fifteen different potentially solid designs, which was

pared down to five for final consideration. These were MARS, RC6, Rijndael, Serpent, and Twofish (clearly naming conventions vary pretty wildly from organization to organization). All of these final five met the design criteria and had no vulnerabilities that could be found over a four year period of intense scrutiny. Near the end of 2000, NIST announced that Rijndael [6] was to be the Advanced Encryption Standard.

So, what is Rijndael, and why did it win? And for that matter, how do you pronounce it? It is an amalgam of the names of its creators, Joan Daemen and Vincent Rijmen. They say they don't care how one pronounces it, as long as it's not "Region Deal". "Rhine Dahl" is suggested. To people that complain about the tongue-twisting, Rijmen replies on his homepage that they are debating the alternate names "Herfstvrucht", "Angstschreeuw", "Koeieuier", and "Bob".

All of that aside, Rijndael is a modification of a cipher that Daemen had developed previously with Lars Knudsen called Square [5]. It was developed with heavy reference to the results that have already been discussed in this paper, particularly the papers of Nyberg [13, 14, 15]. Rijndael can be run in several modes, with varying key and block lengths, with a particular number of rounds defined for each. The form a single round takes is

$$y = L_2(L_1(F(x))) + k. \tag{17}$$

L_1 and L_2 are linear functions that apply diffusion to the input through a method called the **wide-trail strategy** which is beyond the scope of this paper. F is the nonlinear function made up of several identical S-boxes acting on byte-sized inputs, in other words on $\mathrm{GF}(2^8)$. k is the round key. The final round omits the L_2 function. This is a slight variation on the generic round function we have considered up to now, but all of our previous results still apply.

The S-box function $\mathcal{S}$ is what we are most interested in, of course. Depending on the block size, it may be applied in parallel to in the vicinity of two dozen pieces of the intermediate value. The specific criteria for the S-box as given in [6] are as follows:

(1) Invertibility;
(2) Minimization of the largest non-trivial correlation between linear combinations of input bits and linear combination of output bits;
(3) Minimization of the largest non-trivial value in the xor table;
(4) Complexity of its algebraic expression in $\mathrm{GF}(2^8)$;
(5) Simplicity of description.

The first criteria requires $\mathcal{S}$ to be a permutation, as anything else would not be decryptable. The second and third are other ways of stating our **Goals A** and **B**. The fourth is a requirement we have not discussed pertaining to a class of attacks that attempt to create a simple algebraic description of the entire cipher. The fifth is what sets Rijndael apart from the other AES finalists. A clean and simple design means less room for hidden interactions or, worst case, hidden trapdoors.

So what is $\mathcal{S}$? It is the inversion function (12) discussed in Section 11, applied to GF(2^8), with multiplication done under the modulus $x^8+x^4+x^3+x+1$, composed with an invertible affine mapping $p(x) \mapsto q(x)$ given by:

$$q(x) = (x^7 + x^6 + x^2 + x) + p(x)(x^7 + x^6 + x^5 + x^4 + 1) \pmod{x^8 + 1}.$$

Note that the modulus of the affine mapping is not irreducible. The affine function is chosen to force $\mathcal{S}$ as a whole to have a complicated algebraic expression, even though its two component functions are simply described. Also note that this composition does not affect the nonlinearity or the differential uniformity of $\mathcal{S}$.

Proposition 11.1 states that this function will be differentially 4-uniform, and Proposition 11.2 states that the nonlinearity of $\mathcal{S}$ will be greater than or equal to $2^7 - 2^4 = 112$, very close to the upper bound of $2^7 - 2^3 = 120$ proven in Theorem 5.7. These values, combined with the diffusion properties of the linear components of the round, and the number of rounds, makes Rijndael secure against linear and differential cryptanalysis. Neither of these attacks can be leveraged against Rijndael in any way that requires fewer (plaintext, ciphertext) pairs than a full keyspace search. In fact it's not even close. The keyspace itself is large enough to make such a brute-force attack infeasible in any reasonable time frame. A 128 bit key means that, even if we could test one trillion keys per second, it would still take roughly 10^{19} years to be guaranteed key recovery (the age of the universe is frequently estimated to be not much over 10^{10} years). Each additional bit doubles the required time, and the largest key Rijndael supports is 256 bits (well over 10^{57} years required). There have been a number of reduced round attacks proposed on Rijndael since its designation as AES, but none have come close to being feasible against the full cipher in any mode. There is of course new work being done in this area constantly, and a class of attacks called **quadratic cryptanalysis**, or alternately the **XSL attack**, may be applicable at some point in the future. This sort of research is a danger for all ciphers, of course. SHA-1, the dominant hashing algorithm in general use, was broken in February of 2005. There's a looming threat against RSA and several of the discrete-logarithm-based public key ciphers in the form of **Shor's Algorithm**, a method of factoring large composite numbers and finding discrete logs in polynomial time, using (currently) unavailable quantum computers. In other words, nothing is certain, but under the current state of the art, Rijndael is about as secure as it gets.

13. Conclusion

Cryptography is necessary to the functioning of modern society, yet the design of symmetric-key ciphers is with some justification considered a Black Art. Certainly the "blackest" part of such design is the construction of the nonlinear S-box function. Up until very recently, these functions were

designed primarily as random lookup tables with some constraints. With the techniques and rigor advanced by the series of papers by, particularly, Meier and Staffelbach [12], Pieprzyk [16], and Nyberg [13, 14, 15], a new method of designing S-boxes has been advanced, and demonstrated to be effective. The fact that a cipher designed in this way was chosen to be AES out of a field of very qualified candidates indicates that the methods described here will see much more widespread use in the future. Mathematics trumps black magic.

References

[1] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems, in *Journal of Cryptology*, Vol. 4 No. 1 1991.

[2] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. in *Advances in Cryptology: Proceedings of CRYPTO '92*, pp 487-496. Springer Verlag, 1993.

[3] A. Biryukov, C. De Canniere, Linear Cryptanalysis, *Encyclopedia of Cryptogrpahy and Security*, (Kluwer), 2004, to appear.

[4] L. Carlitz and S. Uchiyama, Bounds for Exponential Sums, in *Duke Mathematical Journal* v. 24, 1957, pp. 37-41

[5] J. Daemen, L. Knudsen, and V. Rijmen, The Block Cipher Square, in *Fast Software Encryptions*, E Biham ed., LNCS 1267, SPringer-Verlag, Berlin, 1997

[6] J. Daemen and V Rijmen, AES Proposal: Rijndael, version 2, 1999

[7] J. Gallian, Contemporary Abstract Algebra, 5th edition, Houghton-Mifflin Company, 2002

[8] R. L. Graham, D. E. Knuth, and O. Patashnik, Concrete Mathematics, Addison-Wesley, Reading, MA, 1989.

[9] I. N. Herstein, Topics in Algebra, 2nd edition, John Wiley and Sons, 1975

[10] S. Landau, Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard, *American Mathematical Monthly*, February 2004, pp. 89-117

[11] M. Matsui, Linear cryptanalysis method for DES cipher, in *Advances in Cryptology: Eurocrypt '93*, T. Helleseth, ed., Springer-Verlag, Berlin, 1994, pp. 386-397.

[12] W. Meier and O. Staffelbach, Nonlinearity criteria for cryptographic functions, in *Advances in Cryptology: Eurocrypt '89*, J.-J Quisquater and J. Vandewalle, eds., Springer-Verlag, Berlin, 1989

[13] K. Nyberg, On the construction of highly nonlinear permutations, in *Advances in Cryptology: Eurocrypt '92*, R. Rueppel, ed., Springer-Verlag, Berlin, 1993, pp. 92-98

[14] K. Nyberg, Differentially uniform mappings for cryptography, in *Advances in Cryptology: Eurocrypt '93*, T. Helleseth, ed., Springer-Verlag, Berlin, 1994, pp. 53-64

[15] K. Nyberg and L. R. Knudsen, Provable security against differential cryptanalysis, in *Advances in Cryptology - CRYPTO'92*, vol. 740, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, New York, 1993, pp. 566–574.

[16] J. Pieprzyk, Nonlinearity of exponent permutations, in *Advances in Cryptology: Eurocrypt '89*, J.-J Quisquater and J. Vandewalle, eds., Springer-Verlag, Berlin, 1990, pp. 89-92

Department of Mathematics, Western Washington University, Bellingham, Washington 98226, USA

E-mail address: paul@eigenspace.net