1. AES is weak. 2. Linear time secure cryptography

Warren D. Smith*
warren.wds@gmail.com

May 14, 2007

Abstract — **PRELIMINARY EVALUATION COPY**

The two most famous cryptosystems DES and AES had both been broken by both brute-force (EFF) and linear cryptanalysis (M.Matsui), and by a timing attack (D.J.Bernstein), respectively. We describe a new simple but more powerful form of linear cryptanalysis. As a result, we argue that such cryptosystems as AES, Skipjack, and TEA
(1) are probably breakable [and DES is probably easier to break than so far appreciated],
(2) even if not, they could contain "trapdoors" which would make cryptanalysis far easier than expected for anybody aware of the trapdoor.
If AES's designers had inserted such a trapdoor, it could be very easy for them to convince us of that. But if none exist, then it is probably infeasibly difficult for them to convince us of *that*. (The keys of ciphers with such trapdoors will be determinable from large numbers of known random plaintext-ciphertext pairs, or via a ciphertext-only attack if the plaintext is natural English.)

We then discuss how to use the theory of binary linear error-correcting codes to build cryptosystems provably *not* containing trapdoors of this sort, provably secure against our strengthened form of linear cryptanalysis and against "differential" analysis, *and* immune to Bernstein's timing attack. In particular we prove a fundamental theorem – it is possible to thus-encrypt n bits with security$\geq 2^{cn}$ via an algorithm that takes $\leq \kappa n$ bit-operations, and which also may be implemented in $O(\log n)$ parallel steps each with $O(n)$ work. (Here c and κ are suitable positive constants.) At the end we give tables of useful binary codes.

A cryptosystem has "security level S" if the fastest low-memory cracking algorithm with success probability$\geq 2/3$ performs work roughly equivalent to S encryptions. (Of course, there is a *high*-memory cracking algorithm, which is simply a giant lookup table of the key for every plaintext-ciphertext pair. It would run almost instantaneously.) Any secret key cipher with a K-bit key can be cracked by exhaustive key search by performing $\approx 2^K$ encryptions. It is a usual design aim to try to make the security level attain this 2^K upper bound.

1 DES and AES, their demise, and the demise of privacy generally

DES and its successor AES were the product of cryptosystem-design competitions (1974, 2001) sponsored and judged by the US Government and as such are the two most famous cryptosystems.

DES's obvious weakness was its short (56 bit) secret-key length. That made it vulnerable to brute force key search. Indeed, a DES-cracking engine was built by the Electronic Frontier Foundation in 1998 for under $250,000; it typically cracks DES in under 1 day.

DES also was shown by M.Matsui to be theoretically vulnerable to **linear cryptanalysis.** Matsui [45] argued that DES would be breakable by anybody with access to 2^{45} random plaintext-ciphertext pairs. Matsui then confirmed [46] his theory by implementing and successfully running his attack. The bulk of the time consumption in Matsui's attack was simply producing the plaintext-ciphertext pairs; actually processing them to determine key bits consumed only 20% of the time. Reduced-round versions of DES are attackable faster. Roughly, the number of pairs required grows exponentially with the number of rounds, i.e. please raise it to the power f, $0 < f < 1$, if we are attacking a DES version with only a fraction f of the usual number of rounds. Later, Knudsen & Mathiassen [41] showed that by using *chosen* plaintexts aimed at the specifics of DES's first round, Matsui's attack could be sped up by a factor of ≈ 4, and, e.g. an attack using 2^{42} ciphertexts arising from chosen plaintexts would determine 12 bits of the key with success probability $\approx 86\%$. (At that point the remaining 44 key bits could be determined using a much smaller search; the EFF's DES-cracker could do that in under a minute.) Junod [37] also explored refinements of Matsui's attack, finding 2^{41} DES evaluations would suffice to find the whole DES key with success-probability 85%, and 2^{39} with probability$> 50\%$. He confirmed this by cracking DES 21 times.

After DES's deficiencies become too obvious, **AES** was thrust upon us. AES was designed by Joan Daemen and Vincent Rijmen [17] and in 2001 won a multiyear international cryptosystem design competition run by the USA's NIST (National Standards Institute). It is an elegant and fast design. And Daemen and Rijmen were aware of linear cryptanalysis and specifically designed AES's Sbox to resist it.

*Non-electronic mail to: 21 Shore Oaks Drive, Stony Brook NY 11790.
[1]Who explicitly falsely stated that Sboxes were immune to timing attacks.

However, they were not aware (and neither were the evaluators at NIST[1]) of the fact that every cryptosystem involving table-lookup **Sboxes** is vulnerable to **timing attacks** based on data-dependent cache-miss slowdown behavior for lookup tables, exhibited by modern computers. Daniel J. Bernstein [4] in 2005 successfully mounted an attack on "open AES" software included in "SSL." The software ran on a remote computer with a fixed key, which Bernstein interacted with solely by sending it plaintext, and getting back ciphertexts with timestamps. His attack, which he described as "embarrassingly simple," successfully determined the AES key in 1 day based on 2×10^8 plaintext-ciphertext-time triples[2]

As AES-defenders have observed, this is not really a flaw in the AES *as an abstract algorithm*, but rather in its *implementation*. AES could still be made safe against Bernstein's atack if it were implemented in correctly-designed hardware. However, as Bernstein explains [4], it is extremely difficult or impossible to implement AES on commonly available computers to be both (1) fast and (2) immune to this kind of attack, and (3) even if you do, it could be very hard to be sure that you have, and some person or compiler innocently "optimizing the code" could destroy security, and hence Bernstein concludes (and I agree) that for any cryptosystem purporting to the great generality and applicability that AES does, this is unacceptable! Sadly, almost every cryptosystem so far designed has Sboxes.

But we shall show that AES also has **genuine algorithm weaknesses** that have nothing to do with timing or implementation. First of all we shall argue that AES's supposed resistance to linear cryptanalysis is a myth.

In §2 & §3 we'll outline an attack on AES-256 which plausibly will deduce its key from a number of plaintext-ciphertext pairs well below the claimed security level of 2^{256}. We actually have several claims.

1. We give an analytic method which makes it plausible that a low-memory cracking algorithm exists, which will (with high probability) determine AES keys (or merely reduce the size of the key space by some power of 2) from random plaintext-ciphertext pairs ("pc pairs") and the work and number of pairs required is far smaller than AES-256's alleged security level 2^{256}.
2. *Note:* we do not actually write down this cracking algorithm, and do not know what it is. We merely argue *nonconstructively* that it probably exists. Our analysis (for the first time?) makes it clear that there are really *two* kinds of security level for cryptosystems – security against nonconstructive cracks and security against explicit cracks.
3. Next, even if the above argument if somehow wrong, we still can argue that AES and cryptosystems like it could contain a "trapdoor" intentionally inserted by their designers (Actually, the above argument can be regarded as making it plausible that such a trapdoor exists that is *unintentionally* present. But now we are considering inserting one intentionally.) If so, then anybody aware of this trapdoor could carry out a much faster-than-expected attack against AES. *This* attack would be fully explicit.

The two heuristic assumptions about AES needed to make our cryptanalyses work are

1. The nonlinearities inside AES's Sboxes behave enough like independent random bits.
2. The "codes of the code" for AES (these are certain binary linear error correcting codes that may be associated with secret-key cryptosystems) behave enough like independently selected random binary linear codes.

There is considerable experimental evidence from many workers for many cryptosystems (and also we give some theoretical understanding of why this is true) indicating the validity of assumption #1. But assumption #2 is far less clear.

Recently press reports and lawsuits by the EFF [20] have exposed the fact that the **US government** has mounted a massive wiretapping and databasing effort to get a copy of every email from anybody to anybody and store it forever. Obviously, then, it is willing and able to muster huge attack resources.

A second possible problem with AES – albeit currently nobody really knows how to wield this one effectively (and neither do I) – is the fact that its byte-to-byte Sbox (its sole source of nonlinearity over GF2) is just (aside from some linear transformations) the field-inversion map in $GF(2^8)$. That means every operation AES does, is a *field operation* in the finite field $GF(2^8)$. That means it is subject to **algebraic attacks**.

It would have seemed superior to deny the cryptanalyst the power of field operations and algebraic thinking. For example, the RSA public key cryptosystem is crackable by anybody who can factor large integers, and very sophisticated subexponential-in-N-time algorithms (quadratic sieve, number field sieve, etc) have been developed for prime factorization of N-digit integers. These algorithms are inherently based on the fact that integers are a *ring* and the integers modulo primes form a *field*. Hence, public key cryptosystems based on elliptic curve *groups* [9] seem superior [42]. Because groups offer the cryptanalyst far less to work with than fields, nobody has yet found a way to break these systems in anything below exponential time.

The possibility of algebraic attacks on AES was considered by Schroeppel et al [59]. While admitting they had no effective attack, they demonstrated that the effect (and also the backwards-direction effect) of R AES rounds could be expressed as a continued-fraction-like expression with R decks and 32^R terms. In particular (which they did not point out), their formula shows the following. Suppose you somehow know all bytes of the AES (expanded) key except one. To crack AES, then, you could try all 256 possibilities for the missing byte. If AES were well-designed, we would hope there were no faster way than such exhaustive trial to determine the missing byte. But – is there a faster way? For 2-round AES, setting the 1-round forward formula equal to

[2] Under a year later, Osvik, Shamir, and Tromer [50] gave a more refined attack which cracks AES in only *65 milliseconds* with only 800 pairs! But that required the attack program to be running on the *same* computer in parallel while monitoring timings of both the encryptor and of itself conducting cache-using experiments, which is a rather unrealistic scenario. (If you could do that, why not just listen to the keystrokes of the encryptor?)

the 1-round backward-direction formula shows that the missing byte is the solution of a quartic polynomial equation, and hence is obtainable using the quartic formula without *any* search.[3] This, while not a break of full AES, would seem to suffice to demonstrate at least some kind of suspicion of algebraic weakness.

AES's current status. In view of both (a) Bernstein's timing attack, (b) our nonconstructive argument for a cracking algorithm, (c) the possibility of a "'trapdoor," we believe that

1. AES should be abandoned
2. cryptosystems provably immune to these attacks should be investigated, and
3. our argument for crack-existence, since it is nonrigorous[4], should be investigated more carefully.

2 Linear Cryptanalysis Simplified

Consult the figure. Any cryptosystem (e.g. AES) of the sort we are interested in cryptanalysing here may be written as a "circuit" made of "wires," "XOR gates" (see figure a), and nonlinear multi-input, multi-output "Sboxes." The "input" bits are the key and plaintext, and the output bits are the ciphertext.

Now, as in figure (c), each Sbox may be *replaced* by its best linear approximating circuit (which is made purely of wires, XOR gates, and "constant 1s") *plus* "noise gates" (figure b) attached to the outputs..

Here by "best approximation" we mean, among all Boolean GF2-linear functions of the inputs, find the one which agrees with each output at the most possible input configurations. Put it there, and then add a "noise gate" to that output.

"Noise gates" are 1-input, 1-output gates which, with some probability p (which is smaller, the better the approximation was) perform an inversion $x \to \neg x$, but with probability $1-p$ just transmit the input unaltered. Actually, for this to be an equivalent circuit, these inversion decisions have to be *non*random and in fact are governed by some nonlinear function of the Sbox inputs (or outputs). However, the *approximate probabilistic model* is to model all the noise gates as actually making independent random decisions. That is a good model because cryptosystems are generally intentionally designed to "look random."

Anyway, you can either make this approximation, or not (i.e. be exact). Either way is fine with us for the moment. We shall be able to retain exactness all the way until the final cryptanalytic step.[5] If you want to be approximate, then *label* each noise gate with its characteristic probability value p. In fact it will be more convenient not to deal with p, but rather with the **unbalance** defined by $u = |1-2p|$. Note $0 \le p < 1/2$ and $0 < u \le 1$. If you want to be exact, then *label* each noise gate with its a description of exactly what nonlinear function of what bits tell it when to invert.

Reversibility lemma. It is *artificial* to regard XOR gates as having two "inputs" and one "output." In fact all 3 wires are equivalent; any two of the determine the other's logical state and according to the same function in all cases. Also, is similarly is artificial to regard noise gates as having an "input" and an "output." Again both wires are equivalent and the thing is reversible.

Noise gate mobility lemma. If you slide a noise gate along a wire, through an XOR or other noise gate, and continue along the wire (either wire is fine, if we go through an XOR gate[6]) continuing as far as we want, then the new circuit will be entirely equivalent to the old one. (For an example, see figure e. The top noise gate has slid down to the bottom right, and whether it now is above or below the second gate, does not matter.)

We also remark (although this is not needed for attacking AES) that we also can slide XOR gates along wires and through each other in various ways.

Noise gate unbalance lemma. The unbalance of each noise gate will be at least 2^{-n} assuming all Sboxes have $\le 2n$ input bits (by Rothaus [56], or as in part 6 of our MM theorem in §6). Specific Sboxes may involve even-more-unbalanced noise gates that this (worse Sbox designs have larger u and lead to easier-to-crack cryptosystems) but every noise gate will be at least this unbalanced.

Piling-up lemma: Consider the scenario of figure (f), with n noise-gates in succession along a wire.

1. If the noise-gate inversion decisions are modeled as statistically *independent*, then the net effect is the same as a *single* noise gate with unbalance $U = u_1 u_2 u_3 \cdots u_n$ which is the product of the individual unbalances. (And again $0 < P < 1/2$.)
2. But if two noise-gate inversion decisions perhaps are *dependent* in a positively-correlated manner, then this product is an *lower bound*: $U \ge u_1 u_2 u_3 \cdots u_n$.
3. To get at least constant confidence you know any particular key bit, it suffices to perform U^{-2} experiments (if that bit is attached to an unbalance-U noise gate and you only see the noise-corrupted bit each experiemnt).

[3] Warning: The quartic formula was designed to work over the complex field and will not necessarily work over finite fields because the square and cube roots it asks for, may not exist. (And indeed, polynomial equations over finite fields do not necessarily have solutions, albeit in our cases a solution must exist.) However, square and fourth roots of GF(2^8)-elements always exist and for random field elements, cube roots exist with probability$> 1/3$. So it will work with decent probability.

[4] It is not possible to prove secret-key cryptosystem security without first proving P≠NP. It currently usually is not possible to prove cryptanalyses will be successful without making heuristic probabilistic assumptions about the cryptosystem, which strictly speaking are false. The present paper does not escape from either of these shackles.

[5] Actually, exactness could be retained even then too, but at that point you apparently won't be able to do much useful unless you make an approximation. It may, however, be possible to retain exactness on just *some* noise gates but treating the rest probabilistically, thus improving over a pure-approximate approach. That might be a good focus for future research.

[6] But do *not* "go both ways" – the total number of noise gates is conserved! Also, it is not allowed to slide a noise gate past a "T-junction" where two wires are soldered together, but that need not be an obstacle because it is possible to slide the T-junctions *themselves*, including through XOR gates, provided appropriate duplicate XOR gates are then added to the circuit to generate the correct duplicate signals as in figure g. Or, you can duplicate the noise gate and "slide it both ways" through the T-junction and up both foreign arms. Either way, this costs more gates – but enables further sliding.

Proof. The third claim is standard. To prove the first claim, we first prove it for $n = 2$. The probability the circuit in figure (f) yields a "1" at its output is

$$1/2 - U/2 = P \equiv (1 - p_2)p_1 + (1 - p_1)p_2 = \quad (1)$$
$$= (1/2 - u_2/2)(1/2 + u_1/2) + (1/2 + u_1/2)(1/2 - u_2/2) =$$
$$= 1/2 - u_1 u_2/2.$$

For $n > 2$ the proof is by induction. Now consider the case of dependence. If the two gates are "positively correlated," i.e. one inverting makes it more likely that the other does, then the same derivation as above still works except that the "=" sign which we have written with three bars ($\equiv$) is replaced by "<."

Q.E.D.

This lemma was stated by Matsui. Our only new contribution is the middle claim which largely explains why it is that any real-world departures from the crude model of exact independence, tend to *help* the cryptanalyst. If the cryptosystem uses byte-to-byte Sboxes which permute their 256 inputs (and AES does) then a noise-gate deciding to invert one Sbox output tends to make it *more* likely some other output's noise gate will also invert (if the best-linear approximating map is invertible, i.e. 1-to-1, there has to be another flip or we'll destroy bijectiveness). So case 2 of the lemma tends to apply and linear cryptanalysis ought to work better versus AES than the naive probabilistic model says.

We are now ready to explain **linear cryptanalysis**, Our version of it, which now makes a connection to the theory of linear error correcting codes [43], is this.

Linear cryptanalysis (the algorithm), and the "code of the code":

1. Write down the equivalent circuit of the cryptosystem.
2. Replace all nonlinear Sboxes by their equivalent circuits [best linear approx followed by or preceded by noise gates] as in figure (c). The linear circuits all are to be written in terms of XOR gates and constant 1s only.
3. Use noise-gate mobility to slide all the noise gates along the wires until they reach key-bit inputs. (Note: the topology of the wire-interconnections may make this impossible for some cryptosystems, at least without enormous duplication requirements caused by "sliding through T-junctions" as in footnote 6; but it is *trivially* possible for the AES cryptosystem, since each noise gate (we locate them at the Sbox *inputs*) only needs to slide though a single XOR gate to reach a key-bit, at which point each and every bit of the [extended] key will talk to a totally-GF2-linear circuit through exactly one noise gate.)
4. We may now use reversibility to regard the plaintext and ciphertext bits as the "inputs" and the key bits as "outputs." If the number of key bits is less than or equal to the number of plaintext bits then (in general) the values of the key bits (albeit polluted by noise) will thus be *determined*. We then just do this over and over again with different random plaintext-ciphertext pairs each time to average out the noise and determine the key bits with confidence.
5. However, if the number of key bits is large enough then they will *not* actually be determined by the plaintext and ciphertext (underdetermined system, more unknowns than equations). But each pc-pair (for an n-bit plaintext) will always give a set of n GF2-linear equations satisfied by the set of noise-corrupted-key-bits. We shall describe how to handle that below.

To fix notation, say one of those n linear equations is

$$(K_1 \oplus R_1) \oplus (K_5 \oplus R_5) \oplus (K_7 \oplus R_7) = X \quad (2)$$

where K_j is the jth key-bit, R_j is the "random noise" bit from the noise-gate attached to K_j (which really is not random at all), and $X \in \{0, 1\}$ is the known right hand side for that equation. The *characteristic vector* of this equation is 1000101 (since the 1s are in positions 1,5,7). These characteristic vectors generate (by taking GF2-linear combinations) a *binary linear code* ("the code of the code"). By taking the correct linear combinations (i.e. by performing the correct row-operations on our equations[7]) we can generate *minimum-weight* words in this code, i.e. we can replace our equations by equations that each involve only w terms $(K_j + R_j)$ where w is this code's minimum *Hamming distance*. For example, the sample equation we just wrote has weight $w = 3$. Note that the code of the code depends purely on the circuit-structure of the cryptosystem (using whatever linear approximation schemes we chose) and is *independent* of the particular plaintext and ciphertext we have (that only affects the right hand sides of the linear equations).[8]

Now we may repeatedly do this with different random plaintext-ciphertext pairs each time to average out the noise and determine our min-weight GF2-linear combinations of key bits with confidence. *This constitutes gaining m bits of information about the key, which reduces the cardinality of key space by a factor of 2^m, if the cryptosystem's circuit has m linearly-independent min-weight equations.*

By the piling-up lemma (after moving all w of the R_js to the right hand sides of our min-weight equations) the number T of plaintext-ciphertext pairs you need to gain at least constant confidence of getting any particular linear-combo's Boolean value right, is

$$T = U^{-2w} \quad (3)$$

where U is the unbalance of each noise gate and w is the weight (number of key-bits involved in) that linear combination.

That formula was under the assumption, as in AES, that each key-bit is attached to exactly one noise-gate with unbalance value U (and all these unbalances are the same). Actually, more generally, the unbalances on each bit would *not* necessarily all be equal, in which case we would instead want to find minimum **weighted weight** codewords, i.e. which minimize $\prod U_j^{-2}$ where the U_js are the unbalances of their bits. Then

$$T = \prod U_j^{-2} \quad (4)$$

[7] It is crucial here to note that the "random" R_j bits cancel out, $R_j \oplus R_j = 0$, because they actually are *not* random.

[8] Hence, the job of finding min-weight vectors in this code needs only to be done *once* even if, throughout our future lives, we plan to attack a huge number of pc-pairs and AES instances with different keys. Since we are only arguing *nonconstructively* for the *existence* of a fast low-memory cracking algorithm, we are free to assume foreknowledge of low-weight codewords.

pc-pairs would suffice.

More simply, the cryptanalyst could just ignore such bounds and just keep monitoring the mean and standard deviation of his bit estimates as they update, and just stop once he feels confident enough he really knows the bits (or when he independently verifies that he now knows the key). That way he is not depending on approximate probabilistic models, but instead on experiment.

Finally, let us explain the fact that one can also mount a **ciphertext-only attack** on a cryptosystem this way if the plaintexts are natural English text: there are certain Boolean functions (and hence by Rothaus's approximation theorem, also GF2-linear functions) of natural English text which are naturally unbalanced. For example spaces and the letter "e" are very common, "q" is followed by "u" almost always, etc. Attach such "English detector circuits" to your cryptosystem-circuit's "plaintext input bits" and set the detector outputs to be 1 (really 1⊕noise of course). Now the plaintext bits are no longer "inputs" of the circuit; the only inputs are now the ciphertext bits and 1s, and we can run the usual attack entirely easily. And these attacks all are entirely friendly with special purpose hardware.

3 Cracking AES (and easier if contains trapdoor)

We've now seen that *the complexity of breaking a cryptosystem using linear cryptanalysis depends crucially and in an exponential manner upon the minimum Hamming distance w of the "code of the code."*

But finding – or even merely approximating – the minimum Hamming distance of binary linear codes is, in general, an enormously difficult task. (It is known [48] that even approximating the minimum distance to within a constant factor is not in RP unless RP=NP; and they also have hardness reults even for finding worse-than-constant-factor approximations.) However, it is trivially easy to create a binary linear code which has amazingly low Hamming distance. Upon then presenting that code (in the form of a random generator matrix) to somebody else, they would quite likely experience extreme difficulty in trying to find the small-weight words which you already know.

Let us now consider AES-256's binary code. AES-256 has 14 rounds encrypting a 128-bit chunk of data. Each round XORs 128 key bits with the current data bits, then runs the 16 data bytes through bytewide Sboxes, then performs some GF2-linear operations. (Plus, at the end, there is a final XORing.) The Sbox thus occurs $224 = 14 \times 16$ times in the full AES-256 encryption circuit. This Sbox can be regarded as an 8-bit to 8-bit GF2-linear transformation preceded by 8 "noise gates" that describe its nonlinearity. By taking the right GF2-linear combinations of the 8 Sbox inputs, these noise gates can be made to have unbalance $U = 1/8$. So AES-256 can be regarded as having an associated $[1920, 128, w]$ linear code (where $1920 = 15 \cdot 128$).

Anybody who knew the min-weight codewords in AES's code could crack AES (or at least reduce the size of its keyspace by a power-of-2 factor) with $\approx 64^w$ plaintext-ciphertext pairs, where w is the weight.

The crucial question is *what is w*? We shall discuss three ways to estimate w:

1. An upper bound on w can be deduced from the "linear programming bound" for error correcting codes. Using an approximate (since asymptotic) form of that bound, we find $w \lesssim 790$. Obviously, if the AES-256 code's w really is close to that, then AES-256 would be highly secure against our attack.
2. *Under the approximate assumption* that the AES-256 code behaves like a random code with its parameters, then we can estimate $w \approx 670$. Again, if the AES-256 code's w really is close to that, then AES-256 would be highly secure against our attack.
3. The above two estimates have been pretending there is only *one* "code of the code." Actually, the cryptanalyst's freedom to choose any from an enormous number of combinations of linear-approximations to Sbox outputs means that there are an enormous number of different AES-256 codes. What matters is the *worst* one, i.e. the code which has the *least* w, because that is the one that leads to the most efficient cracking algorithm. We discuss how to estimate *that*, and the indications are that $w_{\text{least}} \approx 1$ is very small indeed, which would imply very efficient ways to crack AES-256.

3.1 Linear programming upper bound on w

The *asymptotic* form of the linear programming bound for binary $[n, k, d]$ error correcting codes (often called the "MRRW bound" [47][9]) is the upper bound in the following. It is valid for all binary codes including nonlinear ones. (The lower bound is the asymptotic form of the Gilbert-Varshamov lower bound and is valid for the *max-d* binary linear code.)

$$1 - H_2(\frac{d}{n}) \lesssim \frac{k}{n} \lesssim \min_{0 \le u \le 1-d/n} 1 + G(u^2) - G(u^2 + 2\frac{d}{n}u + 2\frac{d}{n}) \quad (5)$$

$$\text{where} \quad H_2(p) \stackrel{\text{def}}{=} -p\log_2(p) - (1-p)\log_2(1-p) \quad (6)$$

$$\text{and} \quad G(y) \stackrel{\text{def}}{=} H_2(\frac{1-\sqrt{1-y}}{2}) \quad (7)$$

For the AES-256 code, $n = 1920$, $k = 128$, and we find $d \lesssim 790$.

If AES's code's min-distance w really achieves this upper bound on d, then AES would be very secure against linear cryptanalysis.

3.2 The behavior of random codes

However, since AES's designers never explicitly discussed this issue (and plausibly were unaware of this whole line of thought) it seems more probable that the min-distance of a *random* code with these parameters, is closer to the truth.[10]

In that case our initial guess at w would be slightly below the Gilbert-Varshamov bound.

[9] The simpler "Elias bound" (Theorem 5.2.12 of [?] and theorem 34 in chapter 17.7 of [43]) is $k/n \lesssim 1 = H_2(1/2 - \sqrt{(1/2 - d/n)/2})$. It yields $w \lesssim 873$.

[10] Actually, my guess would be that the AES code actually is *worse* than a random code, but it is only a guess.

The Gilbert-Varshamov bound [66][43] argues that the best (d-maximizing) among a large number of random $[n, k, ?]$ codes must have minimum distance at least d provided

$$\sum_{i=0}^{d-2} \binom{n}{i} \le 2^{n-k} \tag{8}$$

and with $n = 1920$, $k = 128$ this inequality is satisfied when $d = 677$ but not when $d = 678$. (The crossover point, to one decimal, is $d = 677.5$.)

We point out the following reasons why, a priori, we expect w somewhat *below* the GV bound.

1. The Gilbert-Varshamov argument lower-bounds the minimum distance of the *best* of a large set of random linear codes. The *typical* member of this set will have smaller minimum distance than an extreme member. E.g. if we ask the computer to explore random general linear $[62, 30, ?]$, $[93, 30, ?]$, and $[124, 30, ?]$ codes, then we find the empirical distributions of minimum distances in tables 3.1-3.3.

$d =$	4	5	6	7	8	9	10	11
#	0	0	3	19	58	20	0	0

Figure 3.1. The empirical distribution of minimum distances d found in a sample of 100 random $[62, 30, d]$ linear codes. The Gilbert-Varshamov formula assures the existence of a $[62, 30, 10]$ code but not $[62, 30, 11]$. As you can see, the GV bound (which is 10.02 to two decimal places) underestimates the true best-possible minimum distance (≥ 12 according to the table [24] of linear code records) – as it must – but *over*estimates the *typical* minimum distance (7.95) for a random code, and indeed in these experiments the random code never reached the GV bound in 100 trials. ▲

$d =$	14	15	16	17	18	19	20	21
#	0	1	4	32	48	15	0	0

Figure 3.2. The empirical distribution of minimum distances d found in a sample of 100 random $[93, 30, d]$ linear codes. The Gilbert-Varshamov formula assures the existence of a $[93, 30, 19]$ code but not $[93, 30, 20]$. The GV bound (which is 19.7 to one decimal) underestimates the true best-possible minimum distance (≥ 24) – as it must – but *over*estimates the *typical* minimum distance (17.7) for a random code, and indeed in these experiments the random code never exceeded the GV bound in 100 trials. ▲

$d =$	25	26	27	28	29	30	31	32
#	2	8	7	22	40	13	0	0

Figure 3.3. The empirical distribution of minimum distances d found in a sample of 20 random $[124, 30, d]$ linear codes. The GV bound is 30.8; the best currently known code with these parameters has distance 36; and the mean min-distance empirically is 28.5. ▲

2. Pierce [53] showed that asymptotically for large random codes, a probability-fraction tending to 100% of the codes have minimum distance within a factor of $1 + \epsilon$ of the GV bound, for any $\epsilon > 0$.

3. At present, despite 50 years of research by an entire community and an immense number of clever code-constructions, nobody has been able to construct (or nonconstructively prove the existence of) any nonzero- and nonfull-rate family of binary codes which asymptotically beats the Gilbert-Varshamov distance bound arising from random codes by a factor $\ge 1 + \epsilon$ for any $\epsilon > 0$.

4. Finally, remember that really, we are not interested in the min-weight codeword in the AES-256 code (where weight is number of 1-bits in the codeword), but rather we want to minimize the "weighted weight." Because the AES-256 encryption algorithm simply XORs in the final 128 key bits linearly without running them through a nonlinear Sbox, the weights on the final 128 coordinates (which is 1/15 of the coordinates, fractionally) really should be zero because these key bits have no noise-gates attached to them. Hence typically the weighted weight of a codeword will be about 14/15 of its ordinary weight.

3.3 The worst among a large number of random codes

The best (or nonbest, but still good) GF2-linear approximation to a nonlinear Boolean function usually is not unique, and indeed, according to my computer, for *each* of the 510 nonconstant Boolean functions F of AES's Sbox's 8 output bits, there are exactly 5 different GF2-linear functions of its 8 input bits which agree with F in 144 out of the 256 cases (this $p = 144/256 = 9/16$ corresponds to unbalance $U = 1/8$; and the same statement is true with the words "input" and "output" swapped everywhere in this sentence). Further, if we are willing to accept slightly worse linear approximations which agree in only 142/256 cases, then the number "5" grows to "21."

This gives the cryptanalyst tremendous freedom of choice about which linear approximations to use. E.g, since each Sbox has 8 output bits and there are 224 Sboxes, there are $5^{1792} \approx 2^{4161}$ different ways the cryptanalyst could replace those $224 \times 8 = 1792$ bits by their best-possible linear approximations – and if he were willing to employ slightly-suboptimal linear approximations, $21^{1792} \approx 2^{7871}$ ways.

Hence there is not just one "code of the code." Really, AES-256 leads to an enormous number, of order 2^{4161} or much more, of binary codes – and what matters when discussing AES-256's security level against our nonconstructive crack, is the *worst* (w-minimizing) among them. Note that 2^{4161}, and indeed its square root also, both are immensely larger than 2^{1920}.

This makes it seem very plausible that the min-weight in the worst AES-256 code is very small, of order 1. A theorem which supports that is:

THEOREM: Suppose when you sample kz random n-bit words, the expected number of them having weight$\le d$, is ≥ 2. Then the worst (w-minimizing) among z random $[n, k]$ binary linear codes, will (with at least constant probability) have min-distance $w \le 2d$.

The proof is trivial. (This theorem is probably very weak,[11]

[11]The Gilbert-Varshamov bound may be reworded as follows: THEOREM (Gilbert & Varshamov): An $[n, k, d]$ binary linear code exists, if when you sample 2^k random n-bit words, the expected number of them having weight$\le d - 2$, is ≤ 1.

but suffices for our purposes because our z is so enormous.)

So, at least under the crude approximation of the AES-256 codes as independent random codes, we expect $w \approx 1$ and AES-256 should be extremely easy for God's low-memory cracking algorithm to crack.

We warn the reader that this crude approximation *is* crude. (E.g. the AES-codes have similar "macro-structure" and thus are not at all "sampled independently.") However, this has to be regarded as the natural first stab at making such an estimate, and the fact that it leads to such enormous apparent weakness for AES is certainly troubling.

3.4 Now get paranoid

Suppose AES-256's designers had evilly arranged matters so that some "code of the code" (known to them) unexpectedly, contained a remarkably low-weight word (known to them), with say, $w \lesssim 10$. That would constitute a **trapdoor** enabling cracking it with much smaller effort 64^w. If $w = 10$, they could crack AES with $64^{10} = 2^{60}$ pc-pairs. If $w = 7$ then AES-256 would be as easy to crack as the cited attacks on DES (only 2^{42} pc-pairs needed)!

This could be the case even if the random codes approximation underlying our crude analysis in §3.3 was invalid.

3.5 Expanded key versus not – would this really crack AES?

When I said above AES would be "cracked," I meant, more precisely, that a cryptanalyst could discover a high-confidence GF2-linear relation among the 1920 bits of AES's expanded key. But could we determine the entire key in work $\ll 2^{256}$?

The answer is *yes* if enough codes of the code have enough low weight words which are linearly independent enough. In our original attack aimed at getting 128 linear relations, we could have used *another* set of linear approximations to AES's Sboxes. That would be a parallel and somewhat-independent attack of the same sort.

So we could use a second set of approximations to mount a second attack aimed at finding a second set of 128 linear relations. And a third set could be used to derive a third set of 128 linear relations. And so on. If 15 such 128-relation sets were found (or more for safety margin) then we'd end up with enough linear relations – *provided* enough of them turned out to be independent – to simply solve for the entire extended key by solving a 1920-dimensional system of GF2-linear equations (that would take minutes at most on a contemporary computer).

Whether this works depends on how often the minimum Hamming distances of the associated linear codes are small enough often enough (and I have no idea what these min-distances are; the whole attack is predicated on somebody knowing what they are[12]), and how independent the obtained linear relations turn out to be. If this could all be done with say 30 times the work needed for the original 128-relation attack, with $w = 15$ it would be $30 \cdot 2^{90} \approx 2^{95}$ work.

Note that the estimate in §3.3 is so strong that it predicts an enormous number of "codes of the AES code" exist with tiny minweights, so from its point of view, at least, we do not expect any trouble.

4 What now?

Obviously, §2's strengthened form of linear cryptanalysis will attack a wide variety of secret key cryptosystems, not just AES-256, and will quite often raise the worry that there might exist some intentionally or unintentionally inserted "trapdoor." I currently see no feasible way for AES's designers to disprove the existence of this kind of trapdoor. And Bernstein's cache-timing attack [4] seems applicable against essentially any system that employs Sboxes. That's a lot of territory.

In short, *almost every secret key cryptosystem yet proposed is now busted or at least suspect.* We need to design new kinds of cryptosystems immune to both kinds of attack.

To get immunity to Bernstein's timing attack, the obvious fix is to get rid of Sboxes. But, more generally, we also have to get rid of all input-dependent branches, and all operations that are known to take time that varies depending on their data. (For example, on many processors, multiplication, division and/or cyclic shifting by data-dependent amounts take data-dependent runtime, with, e.g. multiplication by -1, 0, 1, or 2 being faster than a general multiplication, or cyclic shift by 0 slots being faster than a general shift.) Bernstein [?] therefore invented `salsa20`, an encryption routine which followed the lead of Helix [21], TEA [57][69], and SHA-1 [?] by employing only $+,-,\wedge,\vee,\oplus,\neg$, and cyclic bitshifts by data-independent amounts. However, it isn't that easy:

1. SHA-1 has been broken: there is an approach to find collisions in only 2^{63} operations as opposed to alleged security level 2^{80} (and indeed explicit collisions have been produced for SHA-1's predecessor SHA-0, Rivest's MD5 hash function, and SHA-1 reduced from 80 to 64 rounds) [68][63][18]. This is despite the fact that SHA-1 endured a multiyear "certification" process.[13]
2. I recommend jettisoning $+$ and $-$ from the operation-repertoire because it seems plausible to me that on some processors (perhaps future ones), these operations will take data-dependent runtime (e.g. faster to compute $x + 0$ than $x + y$). It is known that adding two N-bit numbers necessarily consumes order $\log N$ runtime in a circuit model, which is inferior to wordwide bitwise $\wedge$ and $\vee$ and $\oplus$, which consume only $O(1)$ parallel time.
3. I do not like Bernstein's decision to employ `salsa20` as a pseudorandom number generator i.e. to encrypt data by XORing it with pseudorandom bits forming an artificial "one time pad." One time pads are not secure if they are used twice, and Bernstein's approach makes it too likely that a naive user might do that.

[12] Computing them would be a large computation, but it need only be done once to provide the foundation for an arbitrary number of future attacks on AES – and this "foundation" is entirely storable in a small number of bits. This work in any event is irrelevant because we here are only claiming that a low-space AES-cracking algorithm *exists*, and for that claim it does not matter how hard it is to *determine* that algorithm.

[13] Well, so did AES...

To get immunity to §2's strengthened form of linear cryptanalysis and trapdoor-construction, there are several obvious approaches... but again, a second look reveals that it isn't so easy:

1. Wire your cryptosystem in such a way that noise-gate "sliding" is impeded by lots of "T-junctions." The trouble is... there can be ways to slide past T-junctions anyway (see footnote 6) by modifying the circuit; and how are you going to prove that *no* equivalent circuit is vulnerable? And even if you cannot slide, that does not necessarily imply that (a new form of) linear cryptanalysis cannot still be used. And finally, it seems just as easy to add more nonlinear components to a system as to add more T-junctions, so why not do that instead?
2. Design your cryptosystem's Sboxes to be poorly approximable by linear functions, so that all noise gates have very small unbalances. I see nothing wrong with this idea, but we need a theory of how to build good Sboxes from the small repertoire of allowable primitives – we'll develop one in §6 – and this idea *alone* does not give you as much security as you would like.
3. Wire your cryptosystem in such a way that the "code of the code" has large minimum Hamming distance. The trouble is... there are many possible "codes of the code" arising from different linear-approximation schemes and "sliding" decisions – and how do we assure that they *all* have large minimum distances? Again, a theory is needed, and is provided in §7.

Not surprisingly, both of these theories will rest heavily on the theory of binary linear error correcting codes [43].

5 Some useful gadgets

We shall largely operate on data (plaintext and key) via word-wide $\oplus$, $\vee$, $\wedge$, $\neg$ and cyclic bitshift (by data-independent distances) operations. We need to build useful gizmos out of these.

The *unique* (up to GF2-linear transformations) nonlinear Boolean function of *two* input bits is $\text{AND}(x, y) = x \wedge y$.

If you want a *balanced* nonlinear Boolean function (i.e. its output is equally likely to be 1 or 0 with random input) then we need to use *three* input bits. The "Fredkin gate" (well, up to GF2-linear transformations) is

$$\text{Fred}(x, y, z) = (x \wedge z) \oplus (y \vee z) \tag{9}$$

The "majority function" is

$$\text{Maj}(x, y, z) = (x \wedge y) \vee (y \wedge z) \vee (x \wedge z) = (x \wedge y) \vee (z \wedge (x \vee y)). \tag{10}$$

These are (up to GF2-linear transformations of the 3 inputs and complementation of the 1 output, and ignoring GF2-linear combinations of z with a Boolean function of x, y) the *only* two 3-input nonlinear Boolean functions with output 1-probability exactly equal to 50% [34].

LEMMA [Basic functions]. All three of these gates are predictable by GF2-linear predictors with correctness probability= 3/4, i.e. unbalance= 1/2, and that is best possible. Further, any linear approximant that agrees with Fred (ditto for Maj) on more than 50% of the input configurations, has *balanced error*, that is, when the predictor is wrong, it is equally likely to be a $0 \to 1$ error as a $1 \to 0$ error. All these claims are invariant under invertible GF2-linear transformations.

Proof. In all three cases, just x is a 3/4-predictor. Exhaustive search shows no linear approximant does better than 3/4 (in all three cases) and verifies the balanced error claim for Fred and Maj. There are exactly four better-than-a-coin-toss linear predictors for $\text{Fred}(x, y, z)$, namely x, y, $x \oplus z$, and $y \oplus z$; and also exactly four such predictors, namely x, y, z, and $\overline{x \oplus y \oplus z}$, for $\text{Maj}(x, y, z)$; and also exactly 4 predictors, namely x, y, 0, and $\overline{x \oplus y}$, for $x \wedge y$. All achieve correctness fraction 3/4, i.e. unbalance= 1/2.
Q.E.D.

REMARK ["Differentials"]. The two-input logic gates $x \wedge y$ and $x \vee y$ are "immune to differential analysis" in the sense that, if the 2-bit input word is XORed with some nonzero two bit word Δ, the effect on the output bit is utterly unpredictable from Δ, i.e. it has exactly 50% probability of flipping (assuming all 4 input configurations are equally likely). But that property is *not* enjoyed by Fred, Maj, and GF2-linear functions such as $x \oplus y$: changing all three Maj inputs is guaranteed to flip the output, while changing both x and y is guaranteed to flip $\text{Fred}(x, y, z)$ and not flip $x \oplus y$.

REMARK [Fred versus Maj]. Fred is preferable over Maj because it is easier to compute (3 operations versus 4). However, Fred has the disadvantage for some purposes that, if the z input bit is held fixed, $\text{Fred}(x, y, z)$ is a linear function of the other two inputs. (It is important not to employ nonlinearity only in a bilinear way, e.g. such that for a fixed key the ciphertext is a GF2-linear function of the plaintext. Such a cryptosystem would be trivial to break.)

Here is a useful way to compute an **arbitrary GF2-linear function** of many bits. Suppose the bits you are interested in are the bits in words r_1, r_2,..., r_k at which the masks m_1, m_2,..., m_k respectively have 1 bits. We want to compute F, the XOR of all these bits. Proceed thus. First compute

$$x = (r_1 \wedge m_1) \oplus (r_2 \wedge m_2) \oplus \cdots \oplus (r_k \wedge m_k) \tag{11}$$

and then perform

$$\textbf{for } n = 1 \text{ to } \log_2 W \textbf{ do } x \leftarrow \text{ROT}(x, 2^{n-1}) \oplus x; \textbf{ od};$$

(we assume each word is W bits long and W is a power of 2). The final binary word x will consist of W copies of the same bit, and that bit is the desired value F.

6 How to build highly nonlinear Sboxes from constant-time primitives

The maximization problem confronting us is: how to design Sbox functions with maximum resistance to the attacks we've mentioned, but which are as simple and fast as possible and use only the few and proud constant-time primitive operations we listed to open §5.

In this section we shall show how to construct Sboxes with provably high nonlinearity (small unbalance) from binary linear error correcting codes. Of particular interest are "cyclic," "extended cyclic," and "multicyclic" codes, because they lead to *fast* Sboxes on computers that have "barrel shifter" hardware, and often with no loss of quality because often the best known code parameters happen to be achievable by codes of these types.

6.1 General construction

To construct an Sbox which inputs $2n$ or $3n$ bits, and outputs k bits ($1 \le k < n$) such that every nonconstant GF2-linear function of these k outputs has unbalance U with $0 \le U \le 2^{-d}$, with respect to any GF2-linear combination of the inputs, proceed as follows.

1. Compute n "intermediate bits" by taking the AND and/or OR of pairs of input bits (or Fredkins and/or Majs of triples of input bits, if there are $3n$ inputs).
2. Compute k GF2-linear combinations of these n bits, namely the ones specified by the rows of a $k \times n$ Boolean "generator matrix" for an $[n, k, d]$ binary linear code.
3. These will be the k Sbox outputs.

This construction (using ANDing at the first stage) runs in $n + (d-1)k$ bit operations, each a 2-input logic gate, if each row of the generator matrix achieves the minumum weight d (otherwise somewhat more logic gates will be required). These gates may be arranged so that the number of gate-delays between any input and any output is $\le 1 + \lceil \log_2(d) \rceil$.

Why this works. Observe that any nonconstant GF2-linear function of the k output bits is a GF2-linear combination of the n intermediate bits *with weight (number of bits involved) always at least d.* Each intermediate bit has unbalance$= 1/2$, and all intermediate bits are independent, which in view of the piling up lemma forces unbalance$\le 2^{-d}$.
Q.E.D.

REMARK ["Differentials"]. This construction (assuming we employ AND or OR gates at the initial layer) is "immune to differential analysis" in the sense that, if the $2n$-bit input word is XORed with some nonzero $2n$-bit word Δ, the effect on the k output bits is utterly unpredictable from Δ, i.e. all 2^k possibilities are exactly equally likely for the output XOR-difference.

REMARK ["Avalanching" and the dual code]. If a random generator matrix for the code is used, then this Sbox will usually take $\approx n + kn/2$ bit-operations, and changing a single input bit will with probability $1/2$ cause usually $\approx k/2$ of the k output bits to change, but with probability $1/2$ will change nothing. (The "probability 1/2" is the probability that the intermediate bit changes, assuming all the other input bits are random.) If a low-weight generator matrix is used then we save work (only $\approx n + dk$ bit-operations) but at the cost of reducing the avalanching: now changing a single input bit will with probability $1/2$ cause usually $\approx dk/(2n)$ of the k output bits to change. For some kinds of computation-method, e.g. if based on wordwide operations not bit by bit, this work-savings actually does not exist in which case the former choice is preferable. However, when the work savings *does* exist it is best to take it by using low-weight generators because you'll get more avalanching with less work by using the cheaper Sbox a few more times.

The word "usually" in the last paragraph is unfortunate; we would prefer guarantees. That can be got by using a form of the generator matrix which has at least B one-bits in each column; then changing a single intermediate bit *always* will change at least B output bits. That kind of guarantee often happens effortlessly if we are using, e.g, double-circulant generator matrices. Further, if we change $< d^\perp$ intermediate bits, where $d^\perp$ is the minimum Hamming distance of the *dual* code [43], then no matter which such intermediate-bit-subset is selected, the output will *always* change. For this reason it is desirable to choose codes whose distance and dual distance *both* are large. That happens effortlessly if we use a "formally self dual" rate-1/2 code, for example any "nonsingular pure double circulant" code.

6.2 Linear time secure encryption theorem

THEOREM [Linear time secure encryption]. Choose an $\epsilon > 0$. Define C_m to be the least positive root of $H_2(C_m) = 1/m$ where $H_2(x)$ from EQ 6 is the binary entropy function;

$$C_2 = 0.1100278644\ldots, \quad C_3 = 0.061490470\ldots, \quad \text{and} \quad C_4 = 0.04169269\ldots \tag{12}$$

Then there exists an infinite set S of numbers n, such that there is an Sbox with $4n$ input bits and n output bits such that every nonconstant GF2-linear function of these n outputs has unbalance U with $0 \le U \le 2^{-d}$ with respect to any GF2-linear combination of the inputs, where $d \ge (2C - \epsilon)n$. The sets S, constants C, and amounts of computation needed to use these Sboxes are:

- We may take $S = \{$all sufficiently large multiples of $4\}$, and $C = C_2$ and then if at least some self-dual codes from [44] may be generated by words of asymptotically minimum weight, then the computation takes $\le (2C_2 + \epsilon)n^2$ bit operations; without needing any assumption about weights, $n^2 + O(n)$ bit operations suffice.
- We may take $S = \{$all sufficiently large Artin primes$\}$, and $C = C_2$ and then a circuit exists that implements the Sbox using $T(n)$ bit-operations where $T(n)$ is bounded for $n < 100$ and for larger n we may bound $T(n)$ via a recurrence $T(n) = O(n \log n)T(\log n)$.
- We may take $S = \{$all sufficiently large numbers$\}$, and $C = C_2$ and then under the *assumption* that matrix products (in GF2 arithmetic) of $O(n)$ different matrices that each differ from the identity matrix in $O(1)$ entries, yield some $2n \times 2n$ matrix "random enough" so that its lower half generates a code obeying Pierce's theorem [53], then $O(n)$ bit-operations suffice for an Sbox use. fixed constant κ.
- We may take $S = \{$a certain sequence with bounded gap lengths$\}$, and $C = C_3/6$ (actually, more strongly, one may take $C = 0.015$) and then a polynomial-time constructible family of circuits for all these Sboxes exists. Each circuit implements its Sbox using $\le \kappa n$ bit-operations for some constant κ.

If we instead ask for Sboxes with $8n$ input bits then this is still true (now with $C = C_5^2/540$) but *also* we can make the maximum signal-delay for a bit to pass through all the logic gates in the circuit be $O(\log n)$. Also for Sboxes with $8n$ input bits we can get $S = \{(2^m - 1)2m\}_{m=1,\dots,\infty}$ with $C = C_1/2$ with *double circulant* codes, with polynomial(n) expected construction time. These circuits can be implement with signal-delay $O(\log n)$, or with $O(T(n))$ total logic gates.

We may use this Sbox R times successively (as we will describe in §7) to build an R-round cryptosystem, $R \geq 5$ (recommend $R \geq 10$)[14], which encrypts $5n$ bits using a $(4nR + 6n)$-bit enlarged key and achieves security level$\geq 2^{2d}$ against linear cryptanalytic attacks.

PROOF. Before we begin the proof, we first summarize some known existence results for rate-1/2 binary linear codes. Pierce [53] following Gilbert and Varshamov, showed that for all sufficiently large n, a random $[2n, n]$ binary linear code has minimum Hamming distance d satisfying $C_2 - \epsilon < d/n < C_2 + \epsilon$ for any fixed $\epsilon > 0$ with probability$\to 1$ as $n \to \infty$. MacWilliams, Sloane, and Thompson [44][51] showed that for all sufficiently large n, a $[8n, 4n, d]$ *doubly-even self-dual* binary linear code exists with $C_2 - \epsilon \leq d/(8n) \leq \epsilon + 1/6$. Chen, Peterson, and Weldon [14] showed that if n is an "Artin prime" (i.e. prime such that 2 is a generator of the multiplicative group of integers mod n) then a random *doubly-nonsingular double circulant* binary linear $[2n, n]$ code achieves $d/n \geq C_2 - \epsilon$ with probability$\to 1$. This latter result suffers from the slight flaw that it is a famous open problem in number theory whether there are an infinite number of Artin primes. For practical purposes this flaw is irrelevant because, e.g. 100-digit Artin primes may readily be produced, but for those who care, similar results but not depending on any number-theoretic conjectures were produced by Kasami [39] and Chepyzhov [15]. All of the above were *nonconstructive* existence results – i.e. the algorithms they imply for constructing and verifying the codes that these theorems assure exist in great multitudes, are computationally *expensive*. Constructive existence results – with *cheap* code-constructions and verifications – were found by Sipser and Spielman [60] by using "expander codes." For example, theorem 19 of [60], specialized to the rate-1/2 case, states that there exists a polynomial-time constructible infinite family of $[2n, n, d]$ "expander" codes where $C_4^2 - \epsilon < d/(2n)$. An improved version of their ideas by Barg & Zemor [2] gives (their theorem 14 specialized to rate-1/2) the same result but now with $d/(2n) > 0.026$ (over ten times larger!). Justesen in a famous 1972 paper [38] showed that $[4n, n, d]$ double-circulant codes (but note each circulant has only half of full rank) exist for each n of form $n = (2^m - 1)2m$ with $d/(4n) \geq C_2/2$, and the double-circulant generator matrix is constructible in expected time polynomial in n.

Finally, Guruswami & Indyk [31] found (a weakened form of their theorem 5 specialized to the rate-1/2 case) that that there exists a polynomial-time constructible infinite family of $[2n, n, d]$ binary codes, each of which can be encoded and decoded[15] in $O(n)$ steps, where $C_3/6 - \epsilon < d/(2n)$.

Spielman's [61] linear-time encoding and decoding algorithms (however, these are instead for $[4n, n, d]$ codes with $d/(4n) \geq C_5^2/2160$; these codes also form an polynomial time constructible infinite family) also may be implemented in *parallel* in $O(\log n)$ stages still with only $O(n)$ bit operations total.

The theorem then follows from using the above facts about codes in our above Sbox construction, with the following notes:

1. Multiplying a binary n-bit vector by a Boolean circulant matrix may be done using FFTs or other fast circular convolution algorithms in $O(n \log n)$ arithmetic operations on $O(\log n)$-bit-long numbers. (These "numbers" can be complex with arithmetic being approximate but carried out with enough decimal places that the answer at the end comes out right, or elements of a suitable finite field with exact arithmetic.) These multiplications in turn may be done using fast convolution algorithms.
2. The linear-time encoding algorithms of [31] (or of the earlier [61]) consist of a sequence of matrix-vector multiplications using bit-vectors and Boolean matrices. We start with an n-bit vector and end with a $2n$-bit vector. Because the matrices only differ from the identity matrix on sparse sets, the total number of bit-operations in all these matrix-vector multiplications is $O(n)$. The net effect is equivalent to a single matrix-vector multiplication of a $2n \times n$ Boolean (product) matrix M by a n-bit vector, albeit just doing that directly would take superlinear time. For our purposes in constructing an Sbox, we are not interested in encoding (and especially not interested in decoding[16]). What we want is to multiply M^T by a $2n$-bit vector to get an n-bit output vector. But this may be accomplished by multiplying the vector successively by all the individual matrices in the product defining M, albeit taking the factors in reverse order and transposed.
3. For the parallel log-time claim we need that the number of multiplicand matrices is $O(\log n)$ and that each matrix-vector multiplication may be done in $O(1)$ parallel time because the number of nonzero entries in each row and column is bounded.

The final claim about cryptosystems (not just about Sboxes) will be deferred to §7.

Q.E.D.

This theorem seems the final word at least as far as asymptotic behavior is concerned and aside from the precise values of the constants – obviously linear time is best possible, security $2^{O(n)}$ is then best possible, and logarithmic parallel time also is best possible for any Boolean circuit made of 2-input logic gates whose output depends on all n of its inputs.

However, real-world cryptographers are highly concerned about *non*asymptotic behavior for highly specific input sizes,

[14] With the $8n$-input Sbox described in the final claim, we need $R \geq 9$ with $R \geq 18$ recommended.

[15] Actually their decoding claim only pertains to errors with few-enough erroneous bits – we cannot decode efficiently out to the full code-distance, only to the "design distance" – but that shall be irrelevant for us.

[16] The fact that Spielman et al have efficient decoding algorithms, improving versus naive exponetial-time decoding algorithms, is irrelevant to us. All we need is their fast *en*coding algorithms. Much earlier work – in the 1970s – by Dobrushin et al showed the existence of good code families with linear time encoders.

and they care about keeping constant factors *small*, and programming it on real machines whereupon what matters is not the number of *bit-operations*, but rather the number of *machine instructions*. Most computers can perform word-wide Boolean operations and cyclic shifting of entire words a single instruction, for, e.g. 32-bit-wide words. That leads us to consider...

6.3 Cyclic codes and their friends

Let us now consider cyclic, extended cyclic, and multicyclic binary linear codes. A set of n-bit words is a "cyclic code" if every cyclic shift of every codeword is also a codeword.

There is an elegant theory [43][52][22][16], which unfortunately has never been stated entirely in one place, that makes it possible to find the best cyclic codes much more quickly than a naive exhaustive search. To recapitulate its essentials, codewords correspond to their "generator polynomials" (e.g. $011001 \leftrightarrow x + x^2 + x^5$), and cyclic shifting of an n-bit codeword corresponds to multiplying its polynomial by x modulo $x^n - 1 \bmod 2$. A cyclic linear code is an "ideal" of polynomials modulo $x^n - 1 \bmod 2$ and may be proven to consist exactly of multiples of a *single* "generator polynomial" ("principal ideal"). Then the dimension k of the cyclic code generated by $P(x)$ is $k = n - \deg(P)$. To find all equivalence classes of cyclic codes with n bits it suffices to investigate only the generators that are *factors* (mod 2) of $x^n - 1$ (a tremendous reduction of the search space). Further, given a generator it is possible to find a good lower bound (the "BCH bound" [43][52]) on the min-distance of the cyclic code that it generates. The BCH bound may be evaluated very quickly from the roots in $GF(2^m)$ with $2^{m-1} < n \le 2^m$ of the generator polynomial.[17]

It is also possible to find low-weight codewords (i.e. good upper bounds) more quickly than a naive exhaustive search (see the lemma in [16]). If n is composite, then a theorem of Goethals [22] shows that every cyclic code arises as a "product code" of cyclic codes with n_1 and n_2 bits where $n = n_1 n_2$ and $\mathrm{GCD}(n_1, n_2) = 1$. This code has $k = k_1 k_2$ and $d = d_1 d_2$. Further, the product code of two cyclic codes of relatively prime lengths can always be reinterpreted as a cyclic code. Therefore only prime-power n need to be investigated and generally speaking only odd n lead to good cyclic codes??? SOMETHING IS WRONG. THIS IS NOT TRUE??? SHOULD OMIT OR FIX. Using these techniques, every binary linear code record that arises from a length-n cyclic code with $n \le 130$ has been found [13][54][58], although the reader is warned that (irritatingly) usually the codes themselves were not computed in fully explicit form and it often is not trivial (although the work needed is tiny compared to the work these authors did) to get them.

The "extension" of a linear $[n, k, d]$ code with d odd is the $[n + 1, k, d + 1]$ code that arises by adding an overall parity check bit.

A set of n-bit words is a "quasicyclic code with skip s" if every cyclic shift by s steps of every codeword is also a codeword. For our purposes, quasicyclic codes are best reinterpreted as "multicyclic codes" with s different cycles. We shall be particularly interested in "multicirculant" codes whose $k \times n$ generator matrix consists of $n/k \ge 2$ different $k \times k$ circulant matrices, and we shall be particularly interested in the "multiply nonsingular" case when all of these circulants are invertible over GF2.

6.4 Sboxes from extended cyclic codes

Because most (all?) computers have even wordlengths while cyclic codes with even wordlengths are poor, it is best to focus on *extended* cyclic codes.

A good Sbox which inputs two W-bit words A, B and outputs k bits may be built from an $[n, k, d]$ cyclic code with $n = W - 1$ odd, $1 \le k < n$, as follows. First, compute intermediate bits via $y = A \wedge B$ where $\wedge$ deontes wordwide bitwise ANDing. (One may also employ OR, or with three input words A,B,C one could employ Fred or Maj, and it is permissible to XOR y with any constant word before using it.) Let b be the Wth bit of y. Then compute

$$M = y \oplus \mathrm{ROT}'(y, g_1) \oplus \mathrm{ROT}'(y, g_2) \oplus \cdots \oplus \mathrm{ROT}'(y, g_j) \quad (13)$$

Here $\mathrm{ROT}'(s, v)$ denotes the result of cyclically shifting the $W - 1$ bits of s by v spots in a cycle of length $W - 1$ (the prime indicates the *exclusion* of the Wth bit b) and the gs are integer constants arising as the exponents in the generator polynomial

$$1 + x^{g_1} + x^{g_2} + \cdots + x^{g_j} \quad (14)$$

of the cyclic code modulo $x^n - 1$, where j is even. Now the k output bits are the first k bits of M, *after* each is XORed with b.

This Sbox will achieve unbalance $U = 2^{-d-1}$.

EXAMPLE. Take the $[223, 67, 47]$ cyclic BCH code with generator word

$$\texttt{11AEDBAA E767C497 861C81BE 36955091 F4698719.} \quad (15)$$

Extend it with a parity bit to yield a $[224, 67, 48]$ code. This yields a 448-input 67-output Sbox with all unbalances $U \le 2^{-48}$. It contains 3373 logic gates assuming a min-weight generator exists. There are ≤ 7 gate delays between each input and each output. Also this Sbox may be implemented in software doing W gate operations at once on a machine with W-bit-wide words. The first 64 outputs of this Sbox can be used R times successively (as we will describe in §7) to build an R-round cryptosystem, $R \ge 8$ (recommend $R \ge 16$), which encrypts 512 bits using a $(448R + 576)$-bit enlarged key and achieves security level$\ge 2^{96}$ against linear cryptanalytic attacks.

Note: the "designed distance" 47 is only a lower bound on the true distance d of this BCH code. All we *really* know is $47 \le d \le 74$. If $d > 47$ then better U and security will result than what we said. Indeed, a $[223, 67, 49]$ code is known to exist (albeit the construction is much more complicated – it is not a cyclic code). If that code were used instead then we'd get $U \le 2^{-49}$ and security$\ge 2^{98}$.

[17] Incidentally, BCH-like bounds superior to BCH's original bound but more complicated, are available [65]. These seem to have remained largely unused by the people who keep track of coding theory records. That is unfortunate because probably a large number of new-record distance bounds would result if they were used.

EXAMPLE. Take the $[447, 129, 87]$ cyclic BCH code defined by the generator word (given as 80 hexadecimal digits)

$$\texttt{5A2A239B C51777BB BDD47734 80D7AB7A 429777CC} \qquad (16)$$
$$\texttt{81856D8F DC9CE2C3 42D802A0 F3D9133E 6D321659}.$$

(And again, $d \geq 87$ is only a lower bound.) Extend it with a parity bit to yield a $[448, 129, 88]$ code. This yields a 896-input 129-output Sbox with all unbalances $U \leq 2^{-176}$. It contains 11671 logic gates assuming a min-weight generator exists. There are ≤ 8 gate delays between each input and each output. Also this Sbox may be implemented in software doing W gate operations at once on a machine with W-bit-wide words. The first 128 outputs of this Sbox can be used R times successively (as we will describe in §7) to build an R-round cryptosystem, $R \geq 8$ (recommend $R \geq 16$), which encrypts 1024 bits using a $(896R+1152)$-bit enlarged key and achieves security level$\geq 2^{176}$ against linear cryptanalytic attacks.

6.5 Sboxes from multicyclic codes

We define an Sbox called "MM."

MM Definition. MM first computes $t = A \vee B$, $x = A \wedge B$, $y = C \wedge D$, $z = C \vee D$, and then

$$\begin{aligned} &\mathrm{MM}(A, B, C, D) = \\ &x \oplus \mathrm{ROT}(x, g_1) \oplus \mathrm{ROT}(x, g_2) \oplus \cdots \oplus \mathrm{ROT}(x, g_k) \\ &\oplus y \oplus \mathrm{ROT}(y, h_1) \oplus \mathrm{ROT}(y, h_2) \oplus \cdots \oplus \mathrm{ROT}(y, h_j) \end{aligned} \qquad (17)$$

except that in this formula any subset of the ys may be replaced by zs and also any subset of the xs may be replaced by ts. Here $\mathrm{ROT}(s, v)$ denotes the result of cyclically shifting the W bits of s by v spots and the gs and hs are integer constants with $0 < g_1 < g_2 < \cdots < g_k < W$ and $0 < h_1 < h_2 < \cdots < h_j < W$.

THEOREM [MM Properties].
Let $d \stackrel{\text{def}}{=} j + k + 2$. Suppose the gs and hs are chosen so that a $[2W, W, d]$ linear binary code (i.e. wordlength $2W$, dimension W, and minimum Hamming distance d) is generated by the $W \times 2W$ matrix consisting of a $W \times W$ circulant with 1s in its first row in positions given by $1, g_1, g_2, \ldots, g_k$ and a second $W \times W$ circulant with 1s in its first row in positions given by $1, h_1, h_2, \ldots, h_j$. Also suppose that some subset of the ys (consisting of half of them, "half" meaning more precisely either $\lfloor j/2 \rfloor$ or $\lceil j/2 \rceil$) are replaced by zs and also some subset (also half) of the xs by ts. Finally suppose that the patterns of these replacements are such that any of the 4 thus-defined subsets (perhaps cyclically shifted mod W) intersects any other, (no matter which cyclic shifts mod W are chosen, so long as they aren't the same) in such a way that their symmetric-difference set (elements in exactly one of the two intersecting sets) always has cardinality $\geq c_{ab}$ where the subscripts a and b are each either "g" or "h" and indicate from what kind of generator the two subsets came from.

Then MM has these properties:

1. $\mathrm{MM}(0, 0, 0, 0) = 0$.
2. [Time-reversal symmetry]. $\mathrm{MM}(A, B, C, D) = \mathrm{MM}(B, A, C, D) = \mathrm{MM}(A, B, D, C)$.
3. [Bit counts]. MM is a function of $4W$ input bits that produces W output bits.
4. But each output bit depends on exactly $2d$ input bits.
5. [Nonlinearity]. Let $L(A, B, C, D)$ be a GF2-linear function of the same $4W$ input bits. Any GF2-linear combination of MM's W output bits, is a Boolean-valued function of its $4W$ input bits which disagrees with L in at least a fraction $1/2 - 1/2^{d+1}$ of the 2^{4W} input configurations. (Indeed this will always be the *exact* number of disagreements or agreements.)
6. [Optimality of nonlinearity]. The preceding two claims are best possible; i.e. for *any* Boolean function F of $2d$ input bits, there exists a GF2-linear function L which disagrees with F on exactly a fraction $\leq 1/2 - 1/2^{d+1}$ of their input configurations.
7. [Everything involves everything]. If the underlying double-circulant binary linear code is *doubly nonsingular* (that is, both circulants are invertible matrices over GF2) then every nonconstant GF2-linear function of MM's output bits, involves (when expressed as an XOR of ANDs of pairs of bits) bits from *both* A,B *and* from C,D.
8. [Expansion I]. Changing any single bit of A or of B causes at least $\lfloor k/2 \rfloor$ output bits to change.
9. [Expansion II]. Changing any single bit of C or of D causes at least $\lfloor j/2 \rfloor$ output bits to change.
10. [Expansion III]. Changing any two bits of $A \oplus B$ (by changing exactly two input bits) causes at least c_{gg} output bits to change.
11. [Expansion IV]. Changing any two bits of $C \oplus D$ (by changing exactly two input bits) causes at least c_{hh} output bits to change.
12. [Expansion V]. Changing any one bit of $C \oplus D$ and one bit of $A \oplus B$ (by changing exactly two input bits) causes at least c_{gh} output bits to change.

(end of theorem.)

REMARK [Multicyclic codes]. One could also reach even better properties, at the cost of higher complexity, by using multi-circulant ("quasi-cyclic") codes to build MM functions with more input words. E.g.

$$\begin{aligned} &\mathrm{MM}'(A, B, C, D, E, F) = \\ &w \oplus \mathrm{ROT}(w, f_1) \oplus \mathrm{ROT}(w, f_2) \oplus \cdots \oplus \mathrm{ROT}(w, f_k) \\ &\oplus x \oplus \mathrm{ROT}(x, g_1) \oplus \mathrm{ROT}(x, g_2) \oplus \cdots \oplus \mathrm{ROT}(x, g_k) \\ &\oplus y \oplus \mathrm{ROT}(y, h_1) \oplus \mathrm{ROT}(y, h_2) \oplus \cdots \oplus \mathrm{ROT}(y, h_j), \end{aligned} \qquad (18)$$

where $w = E \wedge F$, would be related similarly to *triple*-circulant codes.

REMARK [Balanced Sboxes]. The theorem's function MM, while fine in many ways, is not exactly "balanced." That traces to the underlying use of the unbalanced nonlinear AND and OR functions in the definitions of t, x, y, z. If you want balance, then instead define

$$x = \mathrm{Fred}(A, B, C), \quad y = \mathrm{Fred}(\mathrm{D}, \mathrm{E}, \mathrm{F}), \qquad (19)$$

say, where now MM is a function of *six* words A, B, C, D, E, F. Then every output bit (and every GF2-linear combination of output bits) will be a balanced nonlinear Boolean function, such that every GF2-linear approximant to it still will have error fraction $\geq 1/2 - 1/2^d$, and further, every GF2-linear approximant that is a better predictor than a fair coin will exhibit *balanced error* as in the lemma in §5.

Proof. Claims 1-3 and 8-12 are immediate consequences of the definitions. Claim 6 is a standard known fact about $2d$-input Boolean functions (functions meeting the bound are called "bent") proven by Rothaus [56], in particular see the Parseval equality argument on his first page. To get claim 5, combine these facts:

1. Bent functions still are bent if their inputs and outputs are pre-transformed by any invertible GF2-linear functions.
2. $F \oplus G$ is a bent function if F and G are bent functions of disjoint sets of input bits, i.e. "with *disjoint* inputs, *no* nonlinearity cancels out."
3. $x \wedge y$ and $x \vee y$ are bent functions of two bits x, y (in fact both are the *same* function up to linears).
4. (Hence by induction) more generally the "vector dot product" $x_1 \wedge y_1 + x_2 \wedge y_2 + \cdots + x_n \wedge y_n$ where any subset of the $\wedge$s may be replaced by $\vee$s, is a bent function of its $2n$ input bits.
5. $(x \wedge y) \oplus (x \wedge y) = 0$, $(x \vee y) \oplus (x \vee y) = 0$, and $(x \wedge y) \oplus (x \vee y) = x \oplus y$ are linear functions of their two input bits, i.e. with the *same* inputs "their nonlinearities exactly cancel out."
6. If a bit x changes, then either $x \vee y$ or $x \wedge y$ has to change.
7. Any linear combination of the W output bits of MM has to be (up to linears) a vector product of two bit-vectors each with $\geq d$ bits, by the definitions of "minimum Hamming distance" and "binary linear code."

Finally, claim 7 is because any linear function of the output bits which does not involve A,B (say) has to correspond to a codeword zero on the coordinates of the first circulant, which by nonsingularity implies the other half of the coordinates must also be zero, which implies the function must be a constant.

Q.E.D.

REMARK [Even split for best expansion]. Best design is to split the d bits approximately evenly between the two types, i.e. have $j \approx k$, because that tends to maximize the expansion factor for a 1-bit alteration of the input to two *composed* MM() applications (due to the theorem that pq is maximized subject to $p + q = d$, $p > 0$, and $q > 0$ if $p = q$).

EXAMPLE. The following C-language code

```
/*Note ^=XOR, &=AND, |=OR, >>=right shift, <<=left*/
/*ROT(X,Y) rotates word x by y slots on 32-bit machine:*/
uint ROT(uint x, uint y){ return (x<<y)|(x>>(32-y)); }

uint MM(uint A, uint B, uint C, uint D){
   uint w,x,y,z;
   w = A|B;  x = A&B;  y = C&D;  z = C|D;
   return w ^ ROT(w,5) ^ ROT(w,7) ^ ROT(x,8) ^
    ROT(x,13) ^ ROT(x,30) ^ ROT(x,31) ^ z ^
    ROT(y,5) ^ ROT(z,10) ^ ROT(y,12) ^ ROT(y,30);
}
```

implements a 128-input 32-output Sbox MM with these properties:

1. Each output bit is a "bent" function of exactly 24 input bits, and each GF2-linear combination of output bits disagrees with each GF2-linear Boolean-valued function of the 128 input bits on at least a fraction $1/2 - 1/2^{13}$ of their 2^{128} configurations (unbalance $U = 2^{-12}$)
2. Each GF2-linear combination of output bits is a function of some bits from *both x and y*.
3. Changing any single bit of A or of B causes at least 3 bits to change in the output.
4. Changing any single bit of C or of D causes at least 2 bits to change in the output.
5. Changing any two bits of $A \oplus B$ (by changing 2 input bits) changes at least 4 output bits.
6. Changing any two bits of $C \oplus D$ (by changing 2 input bits) changes at least 2 output bits.
7. Changing both one bit of $C \oplus D$ and one bit of $A \oplus B$ (by changing 2 input bits) causes at least 3 bits to change in the output.
8. Changing any b bits among the 128 inputs, if $1 \leq b \leq 11$, always causes the output word to change.
9. MM is invariant under exchanging $A \leftrightarrow B$ and/or exchanging $C \leftrightarrow D$.
10. MM's runtime is independent of the data it operates on, so it is immune to timing attacks.

This design was based on the binary linear $[64, 32, 12]$ code with 32×64 generator matrix arising from the following two 32×32 Boolean circulants

$$\{0, 5, 10, 12, 30\} \text{ and } \{0, 5, 7, 8, 13, 30, 31\}. \qquad (20)$$

No three members of the 7-set are congruent mod 32 to an arithmetic progression. Any two cyclic shifts of the 5-set mod 32 have Hamming distance$\geq$ 8. Any two cyclic shifts of the 7-set mod 32 have Hamming distance$\geq$ 8. Any cyclic shift of the 5-set mod 32 has Hamming distance$\geq$ 8 to any cyclic shift of the 7-set.

7 How to build fast cryptosystems quantifiably resistant to linear cryptanalysis

Cryptosystem designs that simply throw a lot of Sboxes together suffers from the problems (1) that they might interact in strange ways including across "round" boundaries, and (2) we can get a tremendous number of "codes of the code." The only way I know to overcome these two problems is to design the cryptosystem such that

1. Even if the cryptanalyst is given "for free," not only plaintext-ciphertext pairs, but in fact *all intermediate results* at each "round boundary," then *still* linear cryptanalysis is infeasibly hard.[18]
2. By designing the Sboxes and/or the way in which they are used carefully, the fact that there are a tremendous number of "codes of the code" won't matter. Essentially, they are all the same code; more precisely they all lead to the same work-lower-bound on $\prod_j U_j^{-2}$.

[18] This approach seems like wasteful overkill that hurts efficiency. But I do not know a less-wasteful idea.

The simplest design is to make each "round" of the cryptosystem just be one big Sbox. Namely, create an Sbox with $(R-1)n$ inputs and n outputs. Then to encrypt an Rn-bit plaintext:

1. Initially, XOR the first n input bits with n key bits.
2. Now perform R rounds. The jth round, $1 \le j \le R$, is as follows.
   (a) Make the $(R-1)n$ inputs of the Sbox be $(R-1)n$ message bits (where we exclude message bits $(j-1)n$ to $jn-1$) XORed with $(R-1)n$ key bits.
   (b) XOR the n Sbox output bits with the excluded messgae bits.
3. After those R rounds, the entire message is encrypted. But to play it safe I would actually recommend doing another R rounds ($2R$ total). Finally, XOR the entire message with Rn more key bits and output it as the ciphertext.

This is the method that proves the last part of the theorem in §6.2. Here R is a constant; in that theorem $R=5$ but by use of Fredkin rather than AND gates at the Sbox's input layer we could make $R=7$, and by use of $[4n,n,d]$ rather than $[2n,n,d]$ codes in the Sbox design we could make $R=9$. $(R-1)n$ key bits are consumed each round, plus the initial/final "wrapper" stages consume in net $(R+1)n$ more key bits.

Note that we still get the same $\prod U$ bound no matter what approximating circuit is used. That is, *every nonvoid approximate GF2 linear relation between Sbox inputs (key and plaintext) and Sbox outputs always involves at least d Sbox-input Fredkin or AND gate outputs; and hence (considering the lemma in §5 and the piling up lemma) no matter what linear approximations are used for each Fredkin or AND gate – and this include the possibility of using several for* each *gate depending on which output bit we want to approximate – any linear approximating circuit must involve noise gates with unbalances U_j such that $\prod_j U_j^{-2} \ge 2^{2d}$.*

This completes the proof of the theorem in §6.2, but: a skeptic might ask "why must we approximate the AND gates at the beginning and then combine our approximations GF2-linearly? Might there be a better approximation circuit?" To answer, the entire Sbox construction in §6.2 is, up to linear equivalence, simply n independent AND gates. Any linear circuit attempting to approximate those AND gates which does not consist of n independent official AND-approximator circuits *must* yield the the same or worse approximation to each of their output bits, and indeed each AND-gate output not approximated by one of the 4 3/4-approximators (or their negations, i.e. 1/4-approximators) must have *exactly* 50% error by the lemma in §5. Now any linear circuit combining those approximations *must* have exactly 50% error on any Sbox output depending on any such unapproximated bit. Any bit with exactly 50% error has unbalnace 0, i.e. causes $\prod_j U_j^{-2} = \infty$.
Q.E.D.

REMARK [Algebraic structure]. With the Sbox construction of §6.1, anybody attempting to deduce the key bits from some plaintext-ciphertext pairs (or to deduce the plaintext bits from some key-ciphertext pairs) will be faced with solving a system of simultaneous quadratic equations over GF2. Although solving systems of *linear* equations is easy (polynomial time) solving systems of *quadratic* equations over the reals is polynomially equivalent in difficulty to solving an arbitrary system of multivariate polynomial equations ("ETR-complete") as you can readily see by considering adding extra varibales for the purpose of reducing polynomial degrees down to 2. Similarly, Cook's famous SAT problem concerning arbitrary boolean circuits is readily re-expressed as a GF2 quadratic equation system solution-existence problem, hence the latter is NP-complete. So this "algebraic structure" seems really not exploitable "structure" at all.

There is another, **more complicated cryptosystem design** that now uses, not one big Sbox, but rather many small Sboxes (say, each producing one machine word worth of output bits and inputting z machine words), each cryptosystem round. The idea is simple. To combine n such small identical Sboxes to produce one big Sbox, apply them all to zn *disjoint* data words to produce n disjoint words. Say the word size is W bits. Now use an $[n,k,d]$ error correcting code as before to linearly combine the most significant bits of each word to get k bits out. Then do the same for the next most significant bit, etc. The net effect is "one big Sbox" that inputs zWn bits and outputs Wk bits. The unbalance of any GF2-linear combination of its output bits is $U \le u^d$ where u upperbounds the unbalances of the output bits of the small Sboxes. Now this big Sbox can be used as before to create a crptosystem.

EXAMPLE. We use the C-code for MM (from the previous example) as the small Sbox. It inputs four 32-bit words and outputs one. We combine these with the $[24,12,8]$ extended Golay code (here regarded as a double circulant code with the first circulant being $I_{12\times12}$ and the second having first row `101111011000`) to produce a "big Sbox" which inputs 96 words `x[0..95]` and outputs 12 words `z[0..11]`, where each word is 32 bits long:

```
BigSbox(uint x[96], uint z[12]){
  int i,j;  uint y[32];
  for(i=0; i<24; i++){/*Intermediate words y[0..23]:*/
    j = i*4;
    y[i] = MM(x[j], x[j+1], x[j+2], x[j+3]);
  }
  for(i=12; i<20; i++){ y[i+12]=y[i]; }/*extra copies*/
  for(i=0; i<12; i++){/*Golay combinations z[0..11]:*/
    z[i] = y[i] ^ y[i+12] ^ y[i+14] ^ y[i+15] ^
        y[i+16] ^ y[i+17] ^ y[i+18] ^ y[i+20];
  }
}
```

The resulting big Sbox has unbalance $U \le 2^{-96}$ where $96 = 8\times12$. It now can be used in the usual manner to produce an R-round cryptosystem ($R \ge 9$ with $R \ge 18$ recommended) encrypting $108 = 96+12$ words, i.e. $108\times32 = 3456$ bits, with security$\ge 2^{192}$. $3072R+3840$ key bits are consumed.

8 Epitaph

Humanity went to considerable effort to devise a very difficult mathematical problem – breaking AES. As we've seen, this effort probably failed.

In the other direction, the creators of the Clay million-dollar problems (e.g. the Riemann hypothesis, the question of existence and uniqueness of the solutions of the Navier Stokes equations, etc.) tried to make important problems that would *not* be too hard to solve. In almost of those cases, they (so far) also failed.

9 Tables of useful binary codes

We first tabulate good cyclic codes of length $n = 2^m - 1$ bits, $15 \le n \le 255$. For cyclic BCH binary linear codes with wordlength $n = 511 = 2^9 - 1$, see [1].

Next we tabulate the best known nonsingular pure double, triple, and quadruple circulant $[n, k, d]$ codes for $k \le 32$, and double circulants ones for k a multiple of 4 up to a few hundred.

Finally we tabulate some miscellaneous good nonsingular pure-multicirculant codes. Some other good ones we also found by Gulliver & Bhargava [25] but unfortunately they did not actually state the code, only its parameters, in most cases. My search program is far-superior and far-inferior to theirs in different regimes. ??? Markus Grassl independently checked the distances on almost all the multi-circulant codes we found.

$[n$	k	$d]$	generator polynomial in hex
7	4	3	D
15	11	3	19
15	7	5	117
15	5	7	765
31	26	3	29
31	21	5	4B7
31	16	7	F5F1
31	11	11	156C8D
31	6	15	3915ED3
63	57	3	61
63	51	5	1395
63	46	7	3B15D [52]
63	39	9	1DDC9B7
63	36	11	C881761
63	30	13	39B5C2CFB
63	28	15	EA3E88C97 [40]
63	24	15	849043596F
63	21	18	5E36C356F89 [52][40]
63	19	19	?? [52][40]
63	18	21	2AF2E52B433D
63	16	23	D4DCBBD0C9B3
63	11	26	?? [52]
63	10	27	2D82AEAD1926B9
63	7	31	1F79D6171B48995
63	3	36	[52]
63	2	42	?? [52]
127	120	3	C1
127	113	5	5F15
127	106	7	360325
127	99	9	1F93D4A3
127	92	11	EAD953887
127	85	13	7B2BE5AF377
127	78	15	33915DDA30D25
127	71	19	11069939123FDA9
127	64	21	F8541A9D18AA212F
127	57	23	4CB2B66FBCDCC4ADC1
127	50	27	3D9DD80E178D643E3225
127	43	31 [40]	17461ECD160E8E8A18D0B3
127	36	35 [24]	D3193DD5793D210D82AC75D
127	29	43	4A392B0318253C32F43ADB045
127	22	47	351D5D6457F297B007AE42F6127
127	15	55	1BAD16CB34E1028393FC18A07BD4D
127	8	63	FDF3D70DD31582F1DB252139688CD5

Figure 9.1. Cyclic codes with wordlength $n = 2^m - 1$.

When $n \le 63$ we give the exact distance for the best of all cyclic codes, from [13][54][58]. The BCH code of designed distance d is the set of polynomials mod $x^n - 1$ mod 2 which have g^1, g^2,..., and g^{d-1} as roots??? For $n = 2^M - 1 \le 255$ the true distance of all BCH codes is known [1][58]. However there are cases [40][1][64] (some shown here, but quite likely not all are known) in which some non-BCH cyclic code is superior to any BCH code.

Any cyclic $[n, k, d]$ code with d *odd* may be extended by adjoining a parity check bit, then shortened by keeping only the codewords with that extra bit= 0, to yield a cyclic $[n, k-1, d+1]$ code; therefore we omit the cyclic codes with even distances that arise in this way.

All of the BCH code generators and BCH designed distances were computed by a computer program written by the author. ▲

$[n$	k	$d]$	generator polynomial in hex
255	247	3	11D
255	239	5	16F63
255	231	7	1BBA1B5
255	223	9	1EE5B42FD
255	215	11	1337DD3AD11
255	207	13	1C7EB85DF3C97
255	199	15	1F36195C443A4E1
255	191	17	16CE707E26B6F9977
255	187	19	157B5976000B493CE9
255	179	21	12CA7239EE08D439812D
255	171	23	1B0E46229C4EE1F8C7319F
255	163	25	1E810DA40F70569BE7529981
255	155	27	1FBE960583ED41BD2A34D37F9B
255	147	29	1D1160F75F3AD55887562C8413C9
255	139	31	13180F68607DB8DE3A5D853CAEEF25
255	131	37	11BCB6CCE6906958AA17F2231050EB39
255	123	39	143182A510D807CF4435A9C614B2EA8CB7
255	115	43	1855B6B7A2029D679E826017CEAB732E75DF
255	107	45	1242FE9A4365732A1EC04EB9E207EBE7A0D921
255	99	47	11AEDBAAE767C497861C81BE36955091F4698719
255	91	51	1BD0B50C35E487AE9E67A9DAA48F6D1F2E8751C971
255	87	53	120BDF38678D3D1D4CE718E7A1321BA5655DB27A2CF
255	85	55	?? [40]
255	79	55	1B7003B9FD7D0020B87221DEC7CC8835ADC9FB585C4ED
255	71	61 [1]	140A722A1A468D36D87A25364E685922A1E56FD1A478C1D
255	63	63 [1]	11EC9E8B4E7646AB351EEFE380F6C49EB4B56F8BD770AC6C1
255	63	65	??? [64]
255	55	63	1D9B1541D04805B06AF58C1A1635618D6F6822DE248B076778F
255	55	70	1A9C665A63B747BB4F4E32088428A547A6F281030493A9479E9 [64]
255	47	85	156EC40F1969AE61B10BFE0C9B3E94DB865930940A360A5AAC67B
255	45	87	6A085C2D4E1C4B241733FA27C1B9EE02938F93EC3682378545361
255	37	91	52F3615A3703C30FF2752C7EB110EEF18F8D38D39F48ECEFC0C4803
255	33	95	[55]
255	29	95	615E6D7B76399AD5C680A78BFC9AE251351027C260C159AF8440EEA1F
255	21	111	55C8D578C1B5AE00FECD787C510D2EE182EAC7962A89ACA38C9B52E77D5
255	13	119	4D0F680A2ABA5922D7BE62A06C046C6FE4B3EB8C0CF9BF45DE162E4C28167
255	9	127	6F582A8F9D4CD021911AB5DA5CC61C9EE8A120F2CA4AFB136CFC5B8EFE9C2F

Figure 9.2. Cyclic binary linear codes with wordlength $n = 255 = 2^8 - 1$. ▲

[n	k	d]	several randomly-chosen example generators
4	2	2	1, 3
6	3	3	3, 5, 6
8	4	4	7, D, B, E
10	5	4	13, B, 16, 19, D, 1C, 1D
12	6	4	17, 1F, 13, 16, 37, 2C, 3A, 25
14	7	4	4F, 56, 62, B, 1C, 6A, 34
16	8	5	47, 53, 65, 71, 2B, 74
18	9	6	6B, 1A5, B3, 4F, 127, 79, 1E4
20	10	6	171, 28D, 7A, 2F9, 2FC, 3C2, 1C3
22	11	7	6D1, 5C5, 476, 1DA, 6E2
24	12	8	7B1, 1BD, 62F, B17, A37
26	13	7	89E, 10D5, 1227, AC3, 1C89, 9E4
28	14	8	1F63, DF6, EB8, 1CAA, 2A72, 2A72, 14FE
30	15	8	6159, 29D8, 1C33, 7AE7, 3FB9, 7988, 3435, 6E5F
32	16	8	C193, BC6C, 3E41, DE86, E93, F680, E176, B258
34	17	8	1E83, 1C398, 30CD, 1253E, 19DC2, 1A45E, 1F353
36	18	8	29C4B, 4EA1, 248B2, 30DA3, 14DD4, 3F973, 3DD54
38	19	8	18D5D, 391A5, 4569B, 5CB09, 27236, 5D756, E31F
40	20	9	D041D, BA16C, 2BEC8, 1A3F1, C468F, 77609, E05E6, 68FB
42	21	10	1D7BEC, 15D676, 1D7BEC, 5F3BF, 16EA57, 1C3BB3, 16F56
44	22	10	C224F, 16C984, 1F3D61, 18E2B5, 34674, 26E255, 2474B3
46	23	11	1BFD4C, 71EC6E, 71DD8D, 6DF255, 2A73ED, 6EE3D8, BCFDA
48	24	12	3D2F57, 7AF92B, B5FC1D, A5EAE7, 3B6BF8, FEB6E, 57787E
50	25	10	1C810C5, 16D700C, 1D5F447, 1A025D, 384706, 4D6EB2, 777612
52	26	10	1059F63, 114B1A3, F0CA2, 2469465, 3B0DD2D, 3C26640, 222C84A
54	27	11	5736508, 57767CB, 3A83DBB, 345496F, 6EE0F9, 4326D73
56	28	12	CF5A902, C29A413, 6EAC225, 86A4A89, 79F32C8, E6AA0A3
58	29	12	F304D2C, EF54BA3, 1C3064BC, D0E17BC, 1DFB891F, 10AFC129
60	30	12	1D2653E4, CC4805D, 3B47EEA, 83F0A90, 156D526, 1D2653E4
62	31	12	30E6045D, 1324DE6, 65BCA9B8, 7AD1543D, 30F1B946, 1957B021
64	32	12	26CC09D4, 2BADE40, 1C8A6F80, 18C01F44,18C01F44, 35BBA78D, 21D8CAB

Figure 9.3. Largest possible min-distance d for a binary linear pure double circulant nonsingular code with block length n, and dimension $k = n/2$. ("Nonsingular" means we assume at least one of the circulants is invertible over GF2; then it may be taken to be the $k \times k$ identity matrix.) Every code listed here is in fact optimal (maximum d) over *all* binary linear codes, i.e. the restriction to pure double circulant nonsingular form does not hurt for $[2n, n]$ codes with $n \le 32$. (I do not know whether it ever hurts; $n = 36$ might be a good candidate for a counterexample.) Found by exhaustive computer search by the author. Essentially the same exhaustive search was also performed independently by Gulliver & Ostergard [29] with the same results. (For the putative next few entries, see [10].) There usually are many codes meeting the bound, so we give at least 5 example generators in each case. However in the cases $k = 8, 16, 18, 22, 24, 28$ the optimal linear code is known to be unique up to isomorphism [29]. The example generators give the right k bits of the n-bit binary word which is the top row of the generator matrix (written in hexadecimal); the omitted left half is $1000\cdots00$ binary. Warning: The generators we give, when written out in full, are not necessarily minimum-weight codewords. ▲

[n	k	d]	code type	example generators in hexadecimal
8	4	4	JxQR S4	7
18	9	6	JxQR	F2
24	12	8	JxQR S4	46F
32	16	8	JxQR S4	7D2
42	21	10	JxQR	581A7
48	24	12	JxQR S4	D5E979
66	33	12	BJKS	two circulants with top rows 235 and 557; & see [26] for SD
68	34	13	BJKS	two circulants with top rows 8CD and BA9
70	35	13	BJKS	two circulants with top rows 697 and E61
72	36	14	BJKS	two circulants with top rows 93B and 3589
74	37	14	JxQR	18F8C90024
76	38	14	BJKS	two circulants with top rows 265 and E7B
78	39	14	BJKS	two circulants with top rows 33B and 4BD
80	40	16	JxQR S4	C573115202
82	41	14	BJKS	two circulants with top rows 26F and 72D
82	41	14	SD	182EF3D3000 [19]
84	42	14	BJKS	two circulants with top rows 26F and 72D
86	43	15	BJKS	two circulants with top rows 149B and 2ECF
86	43	16	SD	7F7101712E2 [19]
88	44	16	SD	37F8B6B, AFA6A7B, 5F7E665, see also [7] for S4
90	45	18	JxQR	missing since no xQR double circulant exists
90	45	16	BJKS	two circulants with top rows 1179 and 3B5F
92	46	16	QC	two circulants with top rows B27 and 23BD
94	46	16	BJKS	two circulants with top rows B27 and 23BD
96	48	16	BJKS	two circulants with top rows A5D and 193F
98	49	16	JxQR	7FBD52AF7F84
100	50	16	BJKS	two circulants with top rows 8AF and 1B9D
104	52	20	JxQR S4	36212E595B3EB, BBE06C7200000 [28]
106	53	16	BJKS	two circulants with top rows 48EB and 56F1
108	54	16	BJKS	two circulants with top rows 36E5 and 4EF3
110	55	16	BJKS	two circulants with top rows 5C9D and 66DF
112	56	?	?	wanted but missing!
114	57	16	JxQR	22F8847D1077DC
118	59	18	Moore	locate 1-bits at squares mod 59 [30][49]
120	60	?	?	wanted but missing!
128	64	20	JxQR S4	F6EE37DF1F21263D
134	67	22	Moore	locate 1-bits at squares mod 67 [30][49]
138	69	22	JxQR	156235414D0DEC2CA
152	76	20	JxQR S4	AA5B7A42497EA3151DD
166	83	16-28	Moore	locate 1-bits at squares mod 83 [30][35]
168	84	24	JxQR S4	(no xQR double circulant exists)
192	96	24 or 28	JxQR S4	DDE6AC8F2CBE2536CBCBBCF
194	97	22, 24, 26, or 28	JxQR	1F4A979E952FF25116AB44527
200	100	32, [64]	JxQR S4	E7D0AA9EAAC9D550257CCAE51
214	107	18-34	Moore	locate 1-bits at squares mod 107 [30][35]
262	131	20-42	Moore	locate 1-bits at squares mod 131 [30][35]

Figure 9.4. Good binary linear pure double circulant codes with block length n, dimension k, and minimum Hamming distance d. BJKS=tailbiting code [10]; JxQR=Jensonized [36] extended QR [43][62] code; bounds on minimum distances from [43] as updated by Boston [11], Coppersmith & Seroussi [16], Grassl [23], and Tjhai & Tomlinson [64].[19] SD=self-dual code; best such code (restricted to pure double circulant) is known for $n \le 88$ from exhaustive searches [32][26][27]. Results for $n > 88$ unfortunately are sparser and arise from nonexhaustive searches. S4 means "doubly even" code (all weights divisible by 4). QC means quasi-cyclic code without self-duality properties. Distance upper bounds are from computer discoveries of low-weight codewords, distance lower bounds are from theorems or exhaustive enumerations of codewords. All these results are from previous authors and are merely copied, not checked, by me. ▲

[19] "Extended quadratic residue codes" have parameters $[n, n/2, d]$ where $n = p+1$ and $p = \pm 1$ mod 8 is prime. These (in their usual presentation) are cyclic codes with circulant generator whose first row is the binary word with 1-bits located at the nonzero quadratic residues mod p. This matrix has rank $(p+1)/2$ not full rank p. Then an extra parity bit is added to each codeword. But a theorem by Jenson [36] is that these (for many but not all p) may be *rewritten* as pure-double-circulant codes.

[n	k	d]	several randomly-chosen example generators (hex)
6	2	4	2,3; 3,1
9	3	4	2,6; 3,7; 1,6; 2,7
12	4	6	D,6; E,C; D,3; C,B
15	5	7	1C,1A; B,E; 16,7; D,19
18	6	8	35,38; 35,7; 35,E; 2E,7; 17,7
21	7	8	71,3B; 2F,61; 6E,42; 54,4D; 35,61
24	8	8	65,8F; 35,5E; 90,E5; 63,8A; 72,31
27	9	10	16F,C5; 16,1B3; 136,1C8; 11E,9C; 183,9B
30	10	10	107,3C6; 391,2EF; 71,34D; 276,322; F1,38E
33	11	11	688,35F; 21F,1A1; 4F7,1D9; 307,24D; 3EE,F5
36	12	12	EE3,D7E; 36F,615; CF6,855; 7C1,D28; 827,6E7
39	13	12	4C6,1523; DFD,CB9; 1737,1D63; 1698,23E; 56B,53A;
42	14	13	1DA5,2585; 8D8,2335; BEB,206F; 121E,239B; 2CA9,C6E
45	15	14	F1C,6115; 58F2,2C5; 393F,793; 386E,6C8C; 7890,7F72
48	16	14	6C31,E2A0; 1379,823D; 1B8F,CC28; 333D,CE62; 1571,4FB6
51	17	16	C15E,C07D; 150E6,29E6; D52C,EC52; EAE0,47E2; B34E,5AD3
54	18	16	238B9,112D5; 2B4A6,2FB48; 3564C,35A81; DE33,587E; 2482F,2C851
57	19	16	72D85,785EA; 526EA,177CA; 4025E,7CCEF; 4D8A,39A9C; 6E34B,2233A
60	20	17	9C70F,A867E; B2350,452C6; 5A4F9,6F4F3; BC390,76AE4; C5ED0,5DCCE
63	21	18	12DB03,182A21; 1DFB4D,1DFDE0; 19B46B,5B3E4; D8466,D79A3; 9D928,14638B
66	22	18	3723BF,1F3CAB; 10A34F,342386; 296CF6,103D9A; BE1B9,2187D5; 1C673D,EF4FA
69	23	19	74201A,2E40EA; 470B92,5442C8; 74AAF,490DAA; 5AC1EA,3C09AA; 6ECED8,4DA555
72	24	20	6B8069,9A21E; F0C872,C20F04; D61CFB,63F36; CCDE17,732F31; 949F15,F754C7
75	25	20	13EC8D0,1D477CF; D0BC7E,1864F00; EA90F3,1F17E90; 17D01EB,DCC93E; 1E9EF9F,1BB5C43
78	26	20	35F5FB6,3AEE445; 2F91218,159A397; 3B92C1B,2156CBA; 13D5272,318B2D8; 1E55DDF,38577C3
81	27	21*	50EDAF6,24FBECF; 5EB168E,54DD201; 68C6EBB,29E6927; 1505F96,66D47C6; 86C9B9,E461B3
84	28	22	8FDDC60,31504BE; 6C2ADCD,A8B1C08; 5B8B400,9D08FB3; 300FF36,C408BD8; 87A8955,3D34AF3
87	29	24	C2EDD0D,13D122F30; [6][5]
90	30	24*	7A398E9,2CE225F6; CB2057B,19AF2E5A; 38F373A1,1AC73EF4; 2CA5713D,1840C5AA
93	31	24	2FCE6FE7,1C1CDA74; 47238AAB,718B675C; 4F42E036,2B198C37; 140EB06B,67163998
96	32	24	68696422,79290BCA; 9839965,7AC16CCF; 1F45032,46C2C5D7; C3BECA6,55EAF9F3; 334BAC2D,150825B8

Figure 9.5. Putative largest possible min-distance d for a binary linear pure triple circulant nonsingular code with block length n, and dimension $k = n/3$. All, except for Bhargava et al's remarkable $[87, 29, 24]$ code which arises from the quadratic residues and nonresidues mod 29, were found by *non*exhaustive computer search by the author. The example generators are the right k and k bits of the n-bit binary word which is the top row of the generator matrix (written in hexadecimal); the omitted left third is $1000\cdots00$ binary. Warning: The generators we give, when written out in full, are not necessarily minimum-weight codewords. Our search rediscovered all previously-known record parameter sets *except* for $[87, 29, 24]$, and found two new records (indicated as *). Also our entries for $k = 20, 23, 24, 25$ improve over [25]'s table of records from 1995. (For the putative next few entries, see [10].) ▲

[n	k	d]	several randomly-chosen example generators (hex)
60	15	20	EE,298B,3A21; 6876,B28,32A8; 430E,4C8,6FE5;
64	16	22	62A0,9B87,82B5; 1EAE,DFA,69A2; 5EF7,D1F2,53E4; A78E,DC0C,3E05
68	17	23	10875,BB19,D841; D65,165F8,D05F; 6B08,3676,149A1;
72	18	24	929E,2042E,306BC; 3AF8A,307B2,4EC8; 259FA,1A873,CEDD; 36D8B,2AB04,33BBB;
76	19	24	3C71,14C71,C07A; D6AF,44283,5E490; 1447C,1C702,4836F; 4BBB4,4577A,5CA76;
80	20	25	2C2BD,AE3F,112E1; C2677,7B31F,B1C09; 71FED,35F32,DFEE6
84	21	27	106260,CE09C,A3AF8; 28C7C,1AEA84,815A2; 1F2122,1C3825,34D14;
88	22	28	14A505,1989,ECF99; 5C7ED,28037D,8EA90; 273C3,308716,17D6D0; 491F,2A2BBF,3B694B
92	23	28	672C04,57234C,668860; 6ABB31,14D5D2,167EBB; 5B25B3,A857C,278B0A
96	24	28	1FFB95,58B722,60C92D; E1DB10,28C51E,69F911; D4CE47,5C44AC,F23546;
100	25	30	DF66B8,363977,1972FD; 13CD999,11E2EBD,16E1386; 3618D7,171EFC8,263631;
104	26	32	5D1450,3ECDD21,173197E; 1E7A4FD,1D37A82,117E233 [10]
108	27	32	4C1C5E6,661B626,341C736; 36D92C2,18B7A7,7F2DD07; 2D0032,4175DD9,47789EF
112	28	32	F8A5B16,D966159,DD6DD77; 98D644C,C4911F7,BF68711; 4578549,87E12BE,8D76679
116	29	33	12DBC9CD,CAAA1AF,A25D73E;
120	30	34	026BBE,380BDF6B,16B611A6; C6D5EAA,187882DF,2F6BBC5A;
124	31	34	57E21B5B,567428B,3E3BD01F; 2D5A1B79,7AB77E2C,29009890; 308FEF36,56577632,1C24562;
128	32	36	667BB4A1,4047883C,1AE54E70; 65458FED,49ECE954,1CA756D6; 2E3D34F3,6979DC51,4E73E14C

Figure 9.6. Putative largest possible min-distance d for a binary linear pure quadruple circulant nonsingular code with block length n, and dimension $k = n/4$. All were found by *non*exhaustive computer search by the author, except for $[104, 26, 32]$ where the second of the two codes we give was found by Markus Grassl. The example generators are the right k, k and k bits of the n-bit binary word which is the top row of the generator matrix (written in hexadecimal); the omitted left third is $1000 \cdots 00$ binary. Warning: The generators we give, when written out in full, are not necessarily minimum-weight codewords. New records starred, and also our entries for $k = 16, 20, 21, 24, 25, 26$ improve improve over [25]'s table of records from 1995. (For the putative next few entries, see [10].) ▲

[n	k	d]	several randomly-chosen example generators (hex)
80	16	28	DB01,F664,213F,3526; C43C,6CFA,3902,2492; 4C04,3D9F,9442,60AE; D805,3A52,EA98,4992
96	16	36	60F1,E29C,EAA5,4898,4698; 47CD,1807,5EA,2CCA,5FF8; 2FCF,EFD,FF2B,5C76,87D5
112	16	42	2533,E8DC,370,BEAD,246D,6732; 5525,10E6,BA4D,51C3,767B,91EB; 7CB2,EEAC,8A78,95A1,8171,EB74
128	16	50	Claimed in [25]
144	16	57	Claimed in [25]
160	16	64	F15F,6CBC,9C76,6D8D,B517,C192,D27C,7F48,5C5A; B240,5679,7517,501E,EEE9,7DBB,42A7,8CB2,F008
176	16	72	F1D4,C118,9023,ACC2,A131,8217,9B5B,982D,95D1,EE7B;
256	16	113	Claimed in [25]
160	32	48*	3EE5ECF4,2D20C360,1732A1A,6FF015E; 20CDC201,794E8F80,3E8604FA,DAEB533

Figure 9.7. Miscellaneous nonsingular quasi-cyclic linear codes $[n, k, d]$ with k a power of 2 generated by $k \times k$ circulants. New record indicated by *. ▲

References

[1] Daniel Augot, Pascale Charpin, and Nicolas Sendrier: Studying the Locator Polynomials of Minimum Weight Codewords of BCH Codes, IEEE Trans. Info. Theory 38,3 (1992) 960-973.

[2] Alexander Barg & Gilles Zemor: Distance properties of expander codes, Distance properties of expander codes, IEEE Trans. Inform. Theory 52,1 (Jan. 2006) 78-90.

[3] D.J. Bernstein: salsa20 page `http://cr.yp.to/snuffle.html`

[4] D.J. Bernstein: Cache timing attack on AES `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`

[5] V.K. Bhargava & C. Nguyen: Circulant codes based on the prime 29, IEEE Trans. Inform. Theory 26,3 (1980) 363-364.

[6] V.K. Bhargava, G.E. Seguin, J.M. Stein: Some (mk, k) cyclic codes in quasi-cyclic form, IEEE Trans. Inform. Theory 24 (1978) 630-632.

[7] V.K. Bhargava & S.E. Tavares: Difference sets of Hadamard type and quasicyclic codes, Info. & Control 26 (1974) 341-350.

[8] Ian F. Blake (ed.): Algebraic coding theory, history and development [annotated reprints of important papers], Dowden, Hutchinson, Ross Inc. Stroudsburg PA 1973.

[9] I.F.Blake, G.Seroussi, N.P.Smart: Elliptic Curves in Cryptography, London Math'l Society Lecture Notes #265, Cambridge University Press 1999. Errata `www.hpl.hp.com/infotheory/errata082900.pdf`.

[10] I.E. Bocharova, P. Johannesson, B.D. Kudryashov, P. Stahl: Tail-biting codes: bounds and search results, IEEE Transactions on Information Theory 48,1 (Jan 2002) 137-148.

[11] Nigel Boston: The minimum distance of the [137, 69] binary quadratic residue code, IEEE Transactions on Information Theory 45,1 (1999) 282.

[12] Anne Canteaut & Fl. Chabaud: A new algorithm for finding minimum-weight words in a linear code: Application to primitive narrow-sense BCH codes of length 511, IEEE Trans. Inform. Theory, 44,1 (1998) 367-378. Preliminary version: Rapport de recherche 2685, INRIA (oct 1995): `ftp://ftp.inria.fr/INRIA/tech-reports/RR/RR-2685.ps.gz`

[13] Chin-Long Chen: Computer results on the minimum distance of some binary cyclic codes IEEE Trans. Info. Theory 16 (1970) 359-360. Only part of Chen's computer output is here; the full output is reprinted as appendix D of [52].

[14] C.L. Chen, W.W. Peterson, E.J. Weldon Jr: Some results on Quasi-Cyclic Codes, Information & Control 15,5 (November 1969) 407-423.

[15] V. Chepyzhov: New Lower Bounds for Minimum Distance of Linear Quasi-Cyclic and Almost Linear Cyclic Codes, Problemy Peredachi Informatsii 28 (January 1992) 33-44.

[16] D.Coppersmith & G.Seroussi: On the minimum distance of some quadratic residue codes, IEEE Transactions on Information Theory 30,2 (1984) 407-411.

[17] Joan Daemen, Steve Borg and Vincent Rijmen: The Design of Rijndael: AES – The Advanced Encryption Standard, Springer 2002. AES is also described in the NIST contest submission document `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf`, [67], and Wikipedia.

[18] Christophe De Canniere & Christian Rechberger: Finding SHA-1 Characteristics: General Results and Applications, In Proceedings of ASIACRYPT 2006 (Springer LNCS #4284) 1-20, Shanghai, China, 3-7 December, 2006.

[19] S.T.Dougherty, T.A.Gulliver, M.Harada: Extremal Binary Self-Dual Codes, IEEE Trans. Info. Theory 43,6 (1997) 2036-2047.

[20] Information about the EFF lawsuit against the USA: `http://www.eff.org/legal/cases/att/`.

[21] Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, Tadayoshi Kohno: Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive, Fast Software Encryption (2003) 330-346 Springer LNCS 2887; also `http://www.macfergus.com/helix`.

[22] J-M. Goethals: Factorization of cyclic codes, IEEE Trans. Inform. Theory 13 (1967) 242-246.

[23] Markus Grassl: On the minimum distance of some quadratic residue codes, ISIT (Int'l Sympos. Info. Theory) 2000 p.253.

[24] Online table of linear code record upper & lower bounds maintained by Markus Grassl `http://www.codetables.de`. Based on earlier table compiled by A.E.Brouwer & T.Verhoeff.

[25] T.A. Gulliver & V.K.Bhargava: An updated table of rate $1/p$ binary quasi-cyclic codes, Appl. Maths. Letters 8,5 (1995) 81-86.

[26] T. Aaron Gulliver & Masaaki Harada: Classification of Extremal Double Circulant Self-Dual Codes of Lengths 64 to 72, Designs Codes and Cryptography 13,3 (1998) 257-269.

[27] T.A. Gulliver & M. Harada: Classification of extremal double circulant self-dual codes of lengths 74-88, Discr. Maths. 306,17 (2006) 2064-2072.

[28] T.A.Gulliver, M.Harada, J-L. Kim: Construction of new extremal self-dual codes, Discr. Maths 263 (2003) 81-91 corrected 289 (2004) 207.

[29] T.A.Gulliver, P.R.J. Ostergard: Binary optimal linear rate 1/2 codes, Discrete Math. 283, 1-3 (2004) 255-261.

[30] T.A. Gulliver & N. Senkevitch: On a Class of Self-Dual Codes Derived from Quadratic Residues, IEEE Trans. Inform. Theory 45,2 (1999) 701-702.

[31] Venkatesan Guruswami, Piotr Indyk: Linear time Encodable/Decodable Codes with Near-Optimal Rate, IEEE Trans. on Info. Theory 51, 10 (Oct. 2005) 3393-3400.

[32] M. Harada, T.A. Gulliver, Hitoshi Kaneta: Classification of extremal double-circulant self-dual codes of lengths up to 62, Discr. Maths. 188, 1-3 (1998) 127-136.

[33] Masaaki Harada, Akihiro Munemasa, Vladimir D. Tonchev: A Characterization of Designs Related to an Extremal Doubly-Even Self-Dual Code of Length 48, Annals of Combinatorics 9 (2005) 189-198.

[34] Michael A. Harrison: On the classification of Boolean functions by the general linear and affine groups, J. Soc. Indust. Appl. Math. 12,2 (1964) 285-299.

[35] T.Helleseth & J.F. Voloch: Double Circulant Quadratic Residue Codes, IEEE Trans. Info. Theory 50,9 (2004) 2154-2155.

[36] Richard A. Jenson: Double circulant construction for quadratic residue codes, IEEE Trans. Info. Theory 26,2 (1980) 223-227.

[37] P. Junod: On the complexity of Matsui's attack, Selected Areas in Cryptography 8 (2001) 199-211, Springer LNCS 2259.

[38] Jorn Justesen: A class of constructive asymptotically good algebraic codes, IEEE Trans. Inform. Theory 18 (1972) 652-656; reprinted in [8].

[39] T. Kasami: A Gilbert-Varshamov bound for quasi-cyclic codes of rate 1/2, IEEE Trans. Inform. Theory 20,5 (1974) 679.

[40] T. Kasami & N. Tokura: Some remarks on BCH bounds and minimum weights of binary primitive BCH codes, IEEE Trans. Inform. Theory 15,3 (May 1969) 408-413.

[41] Lars R. Knudsen, J.E. Mathiassen: A Chosen-Plaintext Linear Attack on DES, A Chosen-Plaintext Linear Attack on DES, (2001) Springer LNCS #1978 = Fast Software Encryption 7 (2000) 262-272.

[42] Arjen Lenstra: Unbelievable Security: Matching AES security using public key systems, pp.67-86 in Asiacrypt 2001.

[43] F.J. MacWilliams and N.J.A. Sloane: The Theory of Error-Correcting Codes, North-Holland, 1977.

[44] F.J. MacWilliams, N.J.A. Sloane, J.G. Thompson: Good self dual codes exist, Discrete Maths 3, 1-3 (1972) 153-162.

[45] M. Matsui: Linear cryptanalysis method for DES cipher, EUROCRYPT (1993) 386-397 Springer LNCS 765.

[46] M.Matsui: The first experimental cryptanalysis of the Data Encryption Standard, CRYPTO 94 (Springer LNCS 839) 1-11.

[47] R.McEliece, E.Rodemich, H.Rumsey, L.Welch: New upper bounds on the rate of a code via the Delsarte-MacWilliams inequalities, IEEE Transactions on Information Theory 23,2 (1977) 157-166.

[48] Ilya Dumer, Daniele Micciancio, Madhu Sudan: Hardness of approximating the minimum distance of a linear code, IEEE Transactions on Information Theory 49,1 (2003) 22-37.

[49] E.H. Moore: Double circulant codes and related algebraic structures, Ph.D. dissertation, Dartmouth College 1976.

[50] Dag Arne Osvik, Adi Shamir, Eran Tromer: Cache attacks and countermeasures: the case of AES, Proc. RSA Conference Cryptographers Track (CT-RSA 2006) 1-20, Springer LNCS 3860 `http://theory.csail.mit.edu/~tromer/papers/cache.pdf`, `http://www.wisdom.weizmann.ac.il/~tromer/papers/cache.pdf`.

[51] Gabriele Nebe, Eric M. Rains, Neil J. A. Sloane: Self-Dual Codes and Invariant Theory, Springer 2006.

[52] W.W. Peterson & E.J. Weldon, Jr: Error-Correcting Codes, 2nd edition, MIT Press: Cambridge, Mass., 1972.

[53] John N. Pierce: Limit distribution of the minimum distance of random linear codes, IEEE Trans. Info. Th. 13,4 (1967) 595-599.

[54] G. Promhouse & S.E.Tavares: The minimum distance of all binary cyclic codes of odd lengths from 69 to 99, IEEE Trans. Info. Theory 24,4 (1978) 438-442.

[55] F. Rodier: On the minimum distance of the duals of 4-error correcting extended BCH codes, IEEE Trans. Information Theory 45,5 (1999) 1677-1678.

[56] O.S.Rothaus: On bent functions, J.Combin.Theory A 20 (1976) 300-305.

[57] Matthew D. Russell: Tinyness: An Overview of TEA and Related Ciphers, `http://www-users.cs.york.ac.uk/~matthew/TEA/`

[58] Dieter Schomaker, Michael Wirtz: On binary cyclic codes of odd lengths from 101 to 127, IEEE Trans. Info. Theory 38,2 (1992) 516-518.

[59] Niels Ferguson, Richard Schroeppel, Doug Whiting: A simple algebraic representation of Rijndael Proceedings of Selected Areas in Cryptography SAC (2001) 113-111 Springer LNCS Lecture Notes in Computer Science #2259. `http://www.macfergus.com/pub/rdalgeq.html`

[60] Michael Sipser & Daniel A. Spielman: Expander Codes, IEEE Transactions on Information Theory 42,6 (1996) 1710-1722.

[61] D.A. Spielman: Linear time encodable and decdable error-correcting codes, IEEE Trans. Info. Thoery 42,6 (1996) 1723-1731.

[62] Ludwig Staiger: On the square-root bound for QR-codes, J. Geometry 31,1/2 (1988) 172-178.

[63] Marc Stevens: HashClash project `http://www.win.tue.nl/hashclash`.

[64] C.Tjhai & M.Tomlinson: Results on binary cyclic codes, Electronics Letters 43,4 (15 Feb 2007) 234-235.

[65] J.M.Van Lint & R.M.Wilson: On the minimum distance of cyclic codes, IEEE Trans. Info. Theory 32 (1986) 23-40.

[66] R.R. Varshamov: Estimate of the number of signals in error correcting codes, Dokl. Acad. Nauk. SSSR 117 (1957) 739-741. In Russian. An English translation appears in [8], but it is better not to read this paper at all and instead see the treatments of Varshamov's theorem in numerous coding theory textbooks.

[67] Neal R. Wagner: The Laws of Cryptography, online book `http://www.cs.utsa.edu/~wagner/lawsbookcolor/laws.pdf`.

[68] Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu: Finding Collisions in the Full SHA-1, Crypto (2005) ??? Later speedup from 2^{69} to only 2^{63} operations to find a collision by was announced at the Crypto 2005 rump session by X.Wang, F.F.Yao, and A.C.Yao. ???

[69] David J. Wheeler and Roger M. Needham: TEA a tiny encryption algorithm, 363-366 in FSE 2 (1994) = Springer LNCS 1008; Correction to xtea, Technical report, Computer Laboratory, University of Cambridge, October 1998.

10 Appendix on Notation

AES = Advanced Encryption Standard.
DES = Data Encryption Standard.
Pc-pair = plaintext-ciphertext pair.

We denote the AND of two binary words x, y bitwise (or x and y could each just be single bits) by $x \wedge y$; OR is $x \vee y$, and XOR is $x \oplus y$. GF(q) denotes the finite field with q elements.

A "linear" function $F()$ is one with the property that

$$F(x + y) + F(0) = F(x) + F(y)$$

where addition is over some appropriate field.

A "binary code" is a set of distinct fixed-length binary words. It is "linear" if whenever two words x and y are in the code, then so is $x \oplus y$. Binary linear codes with 2^k elements may be described concisely by stating their $k \times n$ Boolean "generator matrix" whose rows form a linear basis for it. The entries M_{jk} of a "circulant" $n \times n$ matrix M depend only on $j - k \bmod n$. A binary linear code is "pure double circulant" if its $k \times 2k$ generator matrix consists of two $k \times k$ circulant matrices next to each other. The parameters of a code are $[n, k, d]$ meaning each codeword has length n bits, the there are 2^k codewords, and the minimum Hamming distance between two codewords is d. The "dual code" is the new set of codewords which have dot product zero with any codeword in the original code. Viewed "geometrically," it is an orthogonal subspace, hence also is a linear code.