Cryptography meets voting

Warren D. Smith*
WDSmith@fastmail.fm

September 10, 2005

Abstract — **We survey the contributions of the entire theoretical computer science/cryptography community during 1975-2002 that impact the question of how to run verifiable elections with secret ballots. The approach based on homomorphic encryptions is the most successful; one such scheme is sketched in detail and argued to be feasible to implement. It is explained precisely what these ideas accomplish but also what they do not accomplish, and a short history of election fraud throughout history is included.**

Contents

*21 Shore Oaks Drive, Stony Brook NY 11790.

1 Introduction

We are going to explain, survey, criticize, and evaluate all the main cryptographic procedures that have been proposed for the purpose of holding verifiable and secure secret-ballot elections. We begin by listing election desiderata in §2. Then §3-4 surveys and explains most of the highlights of cryptographic theoretical computer science during 1978-1995, especially "zero knowledge proof" technology. This is all developed from the ground up (or anyhow from a fairly low level) and in enough detail to try to make everything readable by political scientists and programmers, and to permit engineers to begin system implementation now, without need of any source besides this. In fact, this is a superior introduction to mathematical cryptography than any other source I know, although a planned book by Daniel J. Bernstein titled "high speed cryptography" (partially available on his web site) should eclipse us and Schneier's book [136] is a highly recommended broad survey, although limited in its detail and having some astonishing omissions.[1] (Meanwhile, in the other direction, we will point out some political desiderata that seem to have gone unnoticed by the crypto-CS community.) The algorithmic toolkit from §3-4 is summarized in a handy table and then used in §5-7 to design different voting systems.

§6 and §8 review what we have learned. The latter analyses and corrects the adamant anti-electronic-voting views of voting expert Rebecca Mercuri. §9 surveys election frauds throughout history, focusing especially on recent and American history. Due to the timidity of the US press, it is not commonly realized that 3 US presidents during 1950-2000 were elected with substantial aid from fraud, at least comparable to and sometimes far exceeding their winning margins.

Finally, §11 lays out what conclusions we have been able to reach, including some not appreciated before.

The whole *political*-science question of which vote-combining method should be used is largely – but not entirely – independent of the *computer*-science question of how to *implement* a *given* vote-combining method in such a way as to protect voter privacy, make everybody confident the right election results got computed, etc. We are here focusing almost entirely on the computer-science question.

This is a survey of the contributions relevant to voting of the entire CS-cryptographic community. It therefore is mostly *un*original work. Nevertheless, to my surpise it now includes a fair number of new theoretical contributions[2] as well as some numerous improvements more pedagogical rather than foundational. Because there has not previously been a survey of this sort collecting all this material in one place, we are now for the first time able to see the "big picture" and hence to reach some conclusions that seem not to have been previously reached, or at least not previously clearly explained.

2 Election Desiderata

Here are three, possibly conflicting, desires.

1a. Easy cheap elections: To get tremendous savings in cost and increases in accuracy and convenience, we want elections to be run using computers and the internet.

1b. Hard-to-steal: But people are also afraid (with reason!) that such automation would also make it easy to *steal* elections – quite possibly without anybody even noticing! We want it to be difficult or impossible to cheat – so difficult, in fact, that even huge corporations, and spy agencies such as the NSA and CIA, should be unable to do it.

1c. Hack/destruction immunity; recountability: The trouble with running elections via computers, electronics, and the internet is: those things could be destroyed, or rendered temporarily disfunctional, or their data erased, by some enemy. So it is necessary that all votes be stored in lower-tech, but less vulnerable, forms (e.g. on paper ballots) to permit a recount in such an event. But that seems to prevent the cost savings in 1a.

Here is a quadruplet of desires which again seem (now even more strongly) to be in conflict (and also to conflict with 1c):

2a. Secret ballots: Nobody but the voter should know how he voted (because otherwise pressure could be placed on that voter to vote in a certain way).

2b: No sale: Even more strongly, even if the voter *wants* to reveal how he voted, he should be unable to do that in any way more convincing than just his unsupported assertion (because otherwise that voter would be able to "sell his vote"). The voter should still be unable to do this even if he collaborates with a (corrupt) election authority.

2c. Invisible abstention? Some support the still stronger idea is that nobody but the voter (or somebody who has been observing him continually) should even be able to tell *whether* he voted (because otherwise pressure could be placed on that voter to refrain from voting).

2d. Verifiability: All should be able to verify that *only* authorized voters voted, they voted *at most once* and in a *valid* manner, and their votes then were correctly used to determine the election result. Each voter should be able to verify that he successfully voted and his unaltered vote was incorporated into the election result.

2a, 2b, and 2c are really increasing-strength versions of the same thing. We might imagine achieving 2a by having vote submissions be *encrypted* so that nobody besides the voter and recipient knows the vote. With more cleverness perhaps we could make the recipient also incapable of decrypting –

[1] For example, although Schneier extensively discusses Shamir secret sharing (our §4.11), he does not mention many details, e.g. verifiable secret sharing is given only 1 sentence, and ignores its main theoretical use, secure multiparty computation ([18][40], our §4.27).

[2] New tables of nice safeprimes (§3.1), new kinds of signatures (§4.10), new general purpose zero knowledge proof protocols (§4.26), new recognition of the inefficiency of Boudot's interval-membership proofs (§4.22) and first way to repair that flaw, new realizations about voting, and new homomorphic voting scheme involving "designated verifier" zero knowledge proofs to prevent voters from constructing "receipts."

but still able to total the votes! Then if the voter conveniently "forgot" the 300 random bits that he used to produce his encryption, then no coercer would be able to force him to remember them and the vote's privacy would remain secret. But 2b seems much harder – how can we prevent the voter from intentionally remembering, then demonstrating re-creation of his vote encryption to a vote-buyer?

Our final apparent dichotomy:

3a. More powerful computers would make elections and cryptography faster.

3b. More powerful computers make *stealing* elections and *breaking* cryptography faster!

But: are these really incompatible desire-sets? At least in certain idealized mathematical models of the real world, and under certain unproven (but widely believed)[3] assumptions that certain computational problems are super-polynomially difficult, we shall see that these "incompatible" desires actually are simultaneously achievable. The cryptographic ideas that make that possible are extremely ingenious. The goal of this paper is to survey them.

3 The top things to know about crypto

All cryptography exploits the **contrast** between the polynomial and presumed-exponential (or at least, super-polynomial) computational difficulty of performing certain calculations in the "forward" and "reverse" directions. If, say, some forward computation on n bits of data requires $100n^3$ steps but the backward computation requires 2^n steps, then if $n = 100$ the forward computation would take 10^8 steps (i.e. less than one second on a modern machine) and the backward one $2^{100} \approx 10^{30}$ steps (requiring 40,000 years of computing even with all the 10^9 computers in the world working on it in parallel at 10^9 computational steps per second). As computers get faster, the forward computor can employ larger n so that the asymmetry only grows *more* severe. This explains why 3a and 3b are really not in conflict at all.

But it will be much more difficult[4] to reconcile 2a,2b,2c with 1c,2d.

Everything depends on the computational contrast between certain very easy and apparently very difficult tasks. Important tasks nowadays known to be computationally **easy** (i.e. performable in time bounded by a polynomial of N) include [11]:

Arithmetic: Given N-digit numbers a, b, c, compute sum $(a+b)$, difference $(a-b)$, product (ab), remainder after division ($a \bmod b$) and quotient after division $\lfloor a/b \rfloor$, or perform a modular exponentiation ($a^b \bmod c$; see §4.1).

Primality test: Given an N-digit number: decide if it is prime [6][9][51][140].

Finding primes: Find a random N-digit prime number P *along with* the complete factorization of $P-1$ (enabling one to easily find generators g of the multiplicative group modulo P and then to *prove*[5] P prime) [12][97].

Legendre symbol: Compute the Kronecker-Legendre symbol $(a|b)$ where a, b are $(\le N)$-digit integers.

Roots: Compute the rth root $X^{1/r}$ of X modulo some N-digit prime P. (If it exists. Or say so if it does not.)

Extended GCD: [6] Given N-digit numbers a and b compute integers c, r and s so that $ra+sb = c = \mathrm{GCD}(a, b)$. Note this may be used to compute modular inverses: i.e. to compute $a^{-1} \bmod b$ (such that $aa^{-1} = 1 \bmod b$) we may either compute r so that $ra + sb = 1$ if $\mathrm{GCD}(a, b) = 1$ (and then $a^{-1} = r$), or show that $\mathrm{GCD}(a, b) \neq 1$ (in which case no such a^{-1} can exist).

Chinese remaindering: Given n relatively prime numbers M_1, M_2,..., M_n, and given x_1,x_2,..., x_n, compute the unique number Y with $0 \le Y \le \prod_{k=1}^{n} M_k$ and $Y \bmod M_k = x_k$.

Important tasks presently thought to be computationally **difficult** (i.e. for which there apparently is no polynomial time algorithm) include:

Discrete Logarithm: Given N-digit numbers a, b, c, find an integer ℓ so that $a^\ell = b \bmod c$, or prove no such ℓ exists. (Thus if $a = 67$, $b = 63$ and $c = 101$, the answer would be $\ell = 87$ because $67^{87} = 63 \bmod 101$.)

Integer Factoring: Given an N-digit number X, find its smallest divisor greater than 1. (Thus if $X = 165$, the answer would be 3 since $165 = 3 \cdot 5 \cdot 11$.)

[3] Unfortunately, the key embarrassment in Computer Science as of 2004, is that nobody knows how to prove that most "obviously" hard problems actually are hard. The most famous conjecture in computer science is that P≠NP, i.e. that a vast class of problems called "NP-complete" are hard. Everybody believes that but nobody can prove it; the best we can prove is that if *any* problem in NP is too hard to solve in polynomial(n) steps on the hardest n-bit input, then so is *every* NP-complete problem, and further, hundreds of kinds of problems have been shown [70] to be NP-complete. It has also been proven that Discrete Logarithm and some kinds of Quadratic Residuosity problems are "random self reducible" and hence are equally hard on average (i.e. for random input) as they are on worst-case input.

[4] In fact, the vista of cryptography is littered with the bones of those who have published false proofs of schemes for accomplishing this reconciliation. The scheme proposed by Benaloh and Tuinstra in 1994, in the very first paper introducing the idea of "receipt-free' voting (making vote-selling impossible), was shown to be bogus 6 years later [89] by constructing receipts. Then Okamoto in 1996 published another receipt-free voting scheme which he himself later realized allowed receipts. Okamoto published a repaired version [121] in 1997 but heavily employed "anonymous untappable channels," an assumption so strong as to make his scheme nearly useless. Magkos et al in 2001 then proposed another receipt-free scheme now employing tamper-resistant hardware, but it too was flawed [96]. A scheme by Sako and Kilian is still regarded as correct, but after later "clarification" by Michels and Horster was realized to require some rather strong assumptions/restrictions that had not really been explained by its authors; this scheme's later improvement by Hirt and Sako [89] only retains coercion-resistance under the unrealistically strong assumption that the voters *know which* of the tallying authorities are corrupt [96]. The apparently best mixnet scheme by Furukawa and Sako in 2001 was realized later by its authors to be flawed [69]. An important fast-track secret sharing scheme [73] is flawed (their appendix B is insecure). We shall argue in footnote 60 that a widely publicized scheme by Chaum [35] also is unacceptably flawed, and in §7.4 that another scheme by Kiayias & Yung [99] is unacceptably vulnerable to invalid votes. Hopefully the schemes in the present survey now really work as advertised – but I know I made several errors in earlier drafts of this report, and it is difficult to have tremendous confidence in view of this historical record of blunders. Advanced cryptography is a very tricky area.

[5] We define g to be a *generator* mod P if and only if $g^{P-1} = 1 \bmod P$ but $g^{(P-1)/d} \neq 1$ for each prime divisor d of $P-1$. Theorem: $P \ge 3$ is prime if and only if a generator g exists mod P.

[6] GCD stands for Greatest Common Divisor. Thus $\mathrm{GCD}(12, 30) = 6$. The first efficient GCD algorithms were invented by the ancient Greeks.

Quadratic residuosity: Decide whether A is a square modulo M where A, M are $(\leq N)$-digit integers. (For example $24 = 57^2 \bmod 75$ is a square.)

Roots: Compute the rth root $X^{1/r}$ of X modulo some given N-digit integer M. (If it exists. Or say so if it does not.) If M's prime factorization is unknown, this seems hard for each $r \geq 2$.

Notice that factoring is quite similar to being the "reverse" operation of multiplication; in fact if we are multiplying two primes, the two operations are exactly inverse. Further, discrete logarithm and root extractions are quite similar to being the "reverse" operations of modular exponentiation. Producing squares (by squaring) and nonsquares (e.g. by finding A with $(A|M) = -1$, or by multiplying any square by any known nonsquare) modulo M are trivialities, but deciding squarehood and finding square roots both are difficult if M's prime factorization is unknown (although easy if it is known). In all these cases we have forward operations that appear much easier than their reverses.

How hard are these problems? Nobody has ever been able to solve a random discrete logarithm problem with a good-quality 300-digit prime modulus. As of 2001, the record was 120 digits [95], achieved in slightly over 400 MIPS-years of computing.

Nobody has ever been able to factor a 300-digit product of two random nearly-equal primes to back-deduce the primes. As of 2004, the record was 174 digits. This accomplishment required 13,200 MIPS-years of computing with the "general number field sieve" and won a \$10,000 prize from `www.rsasecurity.com/rsalabs`. (A \$20,000 prize remains uncollected for a 640-bit [193-digit] example.)

The two main classes of cryptographic algorithms respectively exploit these two contrasts. Specifically, the RSA cryptosystem and various related ideas, associated with *Rivest, Shamir, and Adleman* (R, S, and A, respectively) seem to be based on the difficulty of integer factoring. A different group of algorithms, associated with the name *Elgamal*, seem[7] to be based on the difficulty of discrete logarithm.[8]

3.1 Essentials of speed, security, and parallelism

In both RSA and Elgamal algorithms, the key forward step, which consumes the most computer time, is usually performing **modular exponentiations** $a^b \bmod c$. Thus, it is important to produce good modular exponentiation software (or, if we *really* want speed, custom *hardware*).

Daniel J. Bernstein has written a highly optimized C program called `Zmodexp0.51` that will compute any 512-bit power modulo any 512-bit integer in at most 1627698 Pentium-II cycles. (In other words, 4.66 milliseconds on a Pentium-II at 350 MHz. Bernstein says it *usually* runs in only 840000 cycles; 1627698 was a worst-case bound.) Although 4.66ms per exponentiation may sound fast, if there are 10^8 voters then any secure-election method needing to perform 1000 exponentiations per voter would require 15 compute-*years* on a Pentium-II/350. Even only 1 modular exponentiation per voter would require 5.4 *days*. (In contrast, it would take only CPU-*seconds* for one such Pentium to add up 10^8 votes, if there were no demands for either verifiability or vote-privacy.) This brings home the need for many algorithms to be *parallelizable* and also makes clear the need for fairly serious computing resources. (With 5000 such Pentiums, costing ≈\$5,000,000, i.e. about \$0.05 per voter, this 15-year runtime would drop to 1 day. Given that our budget were this large, it would probably be worth creating custom hardware for high speed modular exponentiation.)

Known algorithms perform modular exponentiation of N-bit integers in a number of steps bounded by $cN^2 \lg N$, for some constant c. The example of `Zmodexp` suggests that $c \approx 0.69$ if a "step" is a pentium-II cycle.

The reason `Zmodexp` prefers *512-bit* numbers is that it is easiest, on modern computers, to run FFT-based (or Karatsuba-based) "fast multiplication" codes [138][19] on 2^n-bit-long numbers. It is possible to do modular exponentiation even *faster* than `Zmodexp` if especially *nice* moduli are employed. Those who wish to use Elgamal systems thus might want to pick *one* particularly nice prime modulus P, *exactly* 2^n bits long and permitting especially fast computation of $x \bmod P$, and stay with it. For example, we mention the remarkable twin prime $2^{512} - 2^{32} \pm 1$ with a particularly computer-friendly binary form (also $2^{64} - 2^{10} \pm 1$ is another such); the primes $2^{128} - 159$, $2^{256} - 189$, $2^{512} - 569$, $2^{1024} - 105$, $2^{226} - 5$, and the Mersenne primes $2^p - 1$ with p =2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, ...

However, brute speed is not the only game in town. Intelligence also helps. A better top-level design of an algorithm often yields greater speed dividends than merely optimizing its innermost primitive operations.

In particular, in the present case (Elgamal), it does not pay to select a prime modulus P purely to get high modding speeds. We also want to pick P so that we get *high security.* Primes P such that $P - 1$ has only small prime factors are *insecure* because it is possible to solve the discrete logarithm problem within large groups by combining solutions in smaller subgroups [127] using "Chinese remaindering."

So it is instead better to choose P to be a *safeprime*, that is, a prime such that $(P - 1)/2$ is also prime. (Safeprimes are also good choices for the two factors of RSA moduli.)[9] The resulting increase in security then enables a smaller prime to be used, which should lead to a considerable net speed increase even though one will probably be forced to use a prime slightly less congenial to fast modding. Tables 3.1-3.2 give useful safeprimes.

[7] The reason for my use of the weasel-word "seem" is that, although we could break RSA if a fast factorer were available, it is conceivable that there is some way to break RSA even if factoring is infeasibly hard. It would be better if somebody *proved* that factoring hard⟹breaking RSA hard, or equivalently that that breaking RSA easy⟹factoring easy. There in fact have been some successes of this type, for example it has been shown that quadratic residuosity is hard if and only if factoring is hard. Also, breaking even just the least significant bit of the ECC Diffie-Hellman key exchange (§4.6) seems as hard as breaking it entirely (if discrete logarithms are hard) [22]. In this survey we shall ignore all such fine points.

[8] It also has been suggested that some cryptographic feats could be accomplished by exploiting the difficulty of finding the closest lattice point or binary codeword to a given real vector (given a set of generating vectors or codewords for the lattice or binary linear code respectively).

[9] Notice that if P is safeprime, then every nonzero integer mod P, provided it is nonsquare (i.e. half of them), is a generator mod P.

n	a	b	c	d	e
16	269	389	413	473	773
32	209	1409	3509	4577	5453
64	1469	2597	8489	13493	16349
128	15449	21509	40697	43829	68033
256	36113	188069	241457	243017	315053
512	38117	49373	111053	235937	561533
1024	1093337	1370753	1428353	1503509	1833557

Figure 3.1. The five largest safeprimes below 2^n are $2^n - a$, $2^n - b$,..., $2^n - e$. ▲

n	a	$\pm$	b	$\pm$	c
64	14	+	5	+	2
64	19	−	10	−	6
64	20	−	11	−	2
64	21	−	12	−	10
64	23	−	20	−	2
64	24	+	20	−	12
64	24	+	21	−	11
128	24	−	19	−	14
128	24	+	6	−	2
128	26	−	25	+	3
128	27	+	25	+	3
128	29	−	26	+	7
128	33	−	14	−	4
128	33	−	24	+	9
256	24	+	12	−	8
256	25	+	11	−	2
256	26	+	17	+	12
256	35	+	11	+	5
256	38	+	14	+	5
256	46	−	31	−	22
256	49	−	36	−	34
256	51	−	34	−	6
256	64	+	57	−	33
256	64	−	32	+	24
512	40	−	13	−	10
512	46	−	44	−	27
512	48	+	35	−	31
512	50	−	39	−	16
512	58	+	49	+	10
512	61	+	47	+	5
512	64	−	42	−	27
1024	56	+	39	+	8
1024	96	+	69	−	61
1024	101	+	53	+	49
1024	116	−	83	+	23
1024	120	−	86	+	46
1024	121	−	38	+	7

Figure 3.2. Some safeprimes $P = 2^n - 2^a \pm 2^b \pm 2^c - 1$ with nice binary representations. ▲

3.2 Elliptic curve groups – why you want them and how to use them

For this reason (i.e. to get higher security, which in turn leads to the ability to use smaller numbers and hence indirectly to higher speed) it presently seems best to design cryptographic algorithms around *elliptic curve groups* rather than RSA and old-style Elgamal.

A lot of people are frightened of elliptic curves, but after the publication of the excellent book [21] there is no longer any reason for fear. At the core, the situation is quite simple.

Multiplication of nonzero integers modulo a prime P is an *abelian group.* That is, denote $ab \bmod P$ by $a \otimes b$. The "group operation" is $\otimes$, i.e. $1 \otimes a = a \otimes 1 = a$ (identity element), $a \otimes b = b \otimes a$ (commutativity), $a \otimes b \otimes c$ is unambiguous (associativity) and for each a there exists an inverse element a^{-1} so that $a \otimes a^{-1} = 1$. This is in fact a *cyclic* group with $P - 1$ elements. (We shall also write x^r to mean $\underbrace{x \otimes x \otimes \cdots \otimes x}_{r\ x\text{'s in all}}$.)

Now this particular group is actually a *field*, that is, there is also an additive group with a different commutative associative operation $a \oplus b$ (to denote $a + b \bmod P$) with additive identity 0 and inverse operation $\ominus a$ (to denote $P - a$) *coexisting* with it, with $0 \otimes x = 0$ and $(a \oplus b) \otimes c = a \otimes c \oplus b \otimes c$.

Now the important realization is this: Most or all cryptographic algorithms of Elgamal type *only* use the multiplicative group mod P, i..e. only use $\otimes$ and x^{-1} and *never* use $\oplus$ and $\ominus$. I.e. the fact that we actually have a field is wasted on Elgamal. Once we realize that, we realize that these same algorithms may be transplanted into *any* cyclic group, including groups which do *not* arise from fields. The "elliptic curve groups of prime order" are just such cyclic groups, and there are known[10] algorithms by Schoof [139], Elkies, Atkin, and Koblitz enabling us to find nice ones efficiently.

And in fact, transplanting Elgamal algorithms into elliptic curve groups of prime order is a good idea for two reasons:

1. Although the additional mathematical structure inherent in having a field does not help users of Elgamal cryptosystems, it might very well help cryptographers trying to *defeat* those systems.
2. So-called "factor base" or "index calculus" techniques have been used[11] to obtain all the world record largest integer factorizations and discrete logarithm solves. These constitute the only known *sub*exponential-time algorithms[12] for integer factoring and discrete logarithm problems. This whole class of techniques simply cannot be used to attack discrete logarithm problems in appropriately chosen elliptic curve groups [21], and consequently only *exponential*-time algorithms are presently known for solving discrete logarithm problems in most elliptic curve groups.

Hence it seems to be far harder to break the elliptic curve versions of Elgamal cryptosystems. That allows us to use

[10] These algorithms are highly technical. Schoof's main idea is to compute the group order G modulo numerous small primes and then to use chinese remaindering to find it as an integer. Once G is found, anybody may readily confirm its correctness by computing x^G for various random x and confirming that it is always 1.

[11] They form the core of the very successful "number field sieve," "quadratic sieve," "Morrison-Brillhart," and "elliptic curve method" integer-factoring algorithms.

[12] These algorithms have (very roughly) runtimes like $\exp O(\sqrt{N})$ and $\exp O(N^{1/3})$ instead of $\exp O(N)$ (which we are here calling "exponential" time) where N is the total number of digits in all their input numbers. But note that these bounds still grow faster than any polynomial in N.

smaller numbers inside them to get the same level of security. Indeed, [21] estimate that an elliptic curve cryptosystem with a 173-bit-long key, would have the same security as a conventional public key system with a 1024-bit-long key. "Certicom ECC-109 challenges" were solved in 2002 and 2004 (cracking EC cryptosystems with 109-bit keys) to win a $10,000 prize. This effort required 1000s of CPU-years; the ECCp-131 challenge $20,000 prize remains uncollected. Arjen Lenstra [106] estimates that[13]

192-bit AES, 7000-bit RSA, and 384-bit ECC
128-bit AES, 3200-bit RSA, and 256-bit ECC
the second line requiring these approximate processor-cycle counts for an encryption or decryption:
360, 80M, and 1.7M cycles

all have about the same security level[14] against presently known attacks and says "the choice [of ECC] is obvious." Even though performing the basic $a \otimes b$ and x^{-1} operations on elements in elliptic curve groups is more complicated and difficult than performing the corresponding group operations on plain integers modulo P, we still get higher speed because we can use smaller integers. This smallness also enables us to consume less storage space.[15]

There are now five things the reader needs to know about elliptic curve groups:

1. How can we represent their elements inside a computer?
2. What is the identity element we have called 1?
3. How can we perform the $a \otimes b$, x^k, and x^{-1} operations?
4. How can we generate a random group element?
5. Give me a short list of different good quality elliptic curve groups of various suitable large prime orders?

Here are the answers.[16] The *elements* of an elliptic curve group are the 2-tuples (x, y) mod P obeying

$$y^2 = x^3 + ax + b \bmod P, \tag{1}$$

together with one extra point called "the point at infinity" or ∞, which serves as the *identity element* 1. There are approximately P such elements since the right hand side is a square mod P approximately half the time.[17] The *parameters* that specify the group are the prime P, and the integers a, b (which must obey $4a^3 + 27b^2 \neq 0 \bmod P$). An element (x, y) may be represented inside the computer in either of two ways: we may directly state the integers x and y mod P, or we may, more concisely, merely state x and a single bit saying whether or not $2y < P$. Then y is efficiently deducible by computing $y^2 = x^3 + ax + b$ mod P, computing its two square roots y and $P - y$ mod P, and choosing the appropriate one. Note: the point at infinity needs to be represented via some special x-value, for example one such that $x^3 + ax + b$ is nonsquare mod P.

This also leads to a fast way to generate a random group element:

1. Generate a random x mod P, and compute $y^2 = x^3 + ax + b$ mod P.
2. If y is nonsquare (and the x is not the special one representing ∞) then go back to step 1.
3. If x is the special ∞ value, then *with probability* 1/2 output ∞, otherwise go back to step 1.
4. If $y = 0$, then *with probability* 1/2 output $(x, 0)$, otherwise go back to step 1.
5. Compute y^2's two square roots y and $P - y$ mod P, choose one (call *it* y) at random, and output (x, y).

The *group operation* $\otimes$ is then as follows: $\infty \otimes (x, y) = (x, y)$ and $\infty \otimes \infty = \infty$. For finite points with distinct x-coordinates: $(x_1, y_1) \otimes (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = L^2 - x_1 - x_2 \quad \text{and} \quad y_3 = (x_1 - x_3)L - y_1 \tag{2}$$

where

$$L = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } x_1 \neq x_2 \\ (3x_1^2 + a)/(2y_1) & \text{if } x_1 = x_2. \end{cases} \tag{3}$$

The case $x_1 = x_2$ needs to be different to avoid division by 0. There are then exactly two possibilities: either $y_1 = y_2$ (squaring) which is covered in the second case of EQ 3, or $y_1 = -y_2$ in which case we instead use

$$(x, y) \times (x, -y) = \infty. \tag{4}$$

In all of these formulas all arithmetic is done modulo P.

The *inversion* operation is $(x, y)^{-1} = (x, -y)$. Fast powering may be done as in §4.1.[18]

Avoid weak elliptic curves. Elliptic curve groups mod P (P prime) with group order G obeying either $G = P$ [148] or $P^B = \pm 1 \bmod G$ for $1 \leq B \leq 20$ [66] are *weak* and should be avoided. That is, there are abnormally-efficient ways to solve discrete logarithm problems in these groups. Choosing elliptic curves randomly modulo a huge prime P until we get one with prime group order is, of course, extremely unlikely to yield a weak curve. The only reason such curves have arisen in human experience at all is because of too-clever people's attempts to generate anomalously "nice" nonrandom elliptic

[13] K-bit AES-like cryptosystems are suspected to require 2^K effort to crack. This is the same security as an ECC system with a $2K$-bit prime modulus. When K is large, approximately the same security for RSA against the number field sieve is got by using a $0.017K^3/\ln(0.11K^{3/2})^2$-bit composite modulus, which is hugely more expensive.

[14] The latter two are public key systems; the former is a secret key system. Old-style Elgamal should be slightly more secure than RSA.

[15] In fact, let us be clear. Cryptographic algorithms which employ any public key techniques other than elliptic curves, are a *sin* that cost a factor of 50 runtime increase. Our goal throughout this paper will be to avoid that sin.

[16] The reader may painfully confirm that commutativity and associativity follow from our formulas.

[17] **Hasse's theorem:** $P + 1 - 2\sqrt{P} \leq \#\text{points} \leq P + 1 + 2\sqrt{P}$. Thus the number of elements is approximately P. It usually is not *exactly* P, although Miyaji [114] shows how to construct elliptic curve groups in which this exact equality does hold. Miyaji's curves, however, are cryptographically weak [148].

[18] These formulas are not as mysterious as they seem. Associated with any cubic curve C in the xy plane is a natural commutative binary operation: given two points on the curve, draw a line L through them and output the unique third point on $L \cap C$. In the case of *elliptic* curves, a slight modification of this operation miraculously yields a group. The "point at infinity" then obviously serves as the identity element. The $x_1 = x_2$ case is actually a limiting case of the $x_1 \neq x_2$ formula as $x_2 \to x_1$. When we then take all these formulas and set them in a finite field (the integers mod P) instead of the real field, obviously all the identities asserting commutativity, associativity, and so on must still work.

curves, which in too many cases led to anomalously weak ones. Thus Miyaji [114] intentionally generated curves with $G = P$, while some others have generated curves $Y^2 = X^3 + aX + b \bmod P$ with $b = 0$ and $P = 3 \bmod 4$, or with $a = 0$ and $P = 2 \bmod 3$. All three are automatically weak.

A few good elliptic curve groups. The elliptic curve cryptography *standards* documents available from `www.secg.org` tabulate recommended curve parameters; some others are in the FIPS-186 digital signature standard and in appendix A of the superb elliptic curve crypto book [21] (which also discusses all of these topics in far greater detail).

A good toy example curve is $Y^2 = X^3 - 3X + 7 \bmod 10007$, which has 10193 points on it (including ∞). Here $P = 10007$ and $G = 10193$ both are prime. Of course this is far too small for any cryptographic use. Serious cryptographic curves are tabulated in table 3.3.

secp128r1 standard curve: $P = 2^{128} - 2^{97} - 1, \quad a = -3,$ $b = \texttt{E87579C1 1079F43D D824993C 2CEE5ED3},$ $G = 2^{128} - 2^{97} + \texttt{75A30D1B 9038A115}.$
WDS's first 128-bit curve: $P = 2^{128} - 2^7 - 2^5 + 1 = 2^{128} - 159, \quad a = -3, \quad b = 63,$ $G = 2^{128} + \texttt{1 58CEDD4E 48CEA415}.$
WDS's second 128-bit curve: $P = 2^{128} - 2^{18} - 1, \quad a = -3, \quad b = 131,$ $G = 2^{128} + \texttt{1 39A8A6A8 FE09646D}.$
secp192r1=NIST P-192 standard curve: $P = 2^{192} - 2^{64} - 1, \quad a = -3,$ $b = \texttt{64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1},$ $G = 2^{192} - \texttt{662107C9 EB94364E 4B2DD7CF}.$
WDS's 192-bit curve: same as above but $b = 446, \quad G = 2^{192} - \texttt{1 B7A36C38 1E019CD8 01A11015}.$
secp224r1=NIST P-224 standard curve: $P = 2^{224} - 2^{96} + 1, \quad a = -3,$ $b = \texttt{B4050A85 0C04B3AB F5413256 5044B0B7 D7BFD8BA 270B3943 2355FFB4},$ $G = 2^{224} - \texttt{E95D 1F470FC1 EC22D6BA A3A3D5C3}.$
secp256k1 standard curve: $P = 2^{256} - 2^{32} - \texttt{3D1}, \quad a = 0, \quad b = 7,$ $G = 2^{256} - \texttt{1 45512319 50B75FC4 402DA173 2FC9BEBF}.$
WDS's 256-bit curve: previous P but $a = -3, \ b = -109,$ $G = 2^{256} - \texttt{1 84CC553B 62923160 34742BA5 066C2CE1}.$

Figure 3.3. Some cryptographically useful elliptic curves $Y^2 = X^3 + aX + b \bmod P$ with P prime. Each curve has group order G, i.e. G =number of points $\infty \cup (X, Y) \bmod P$, which is *prime*. Numbers in this `font` are in radix-16, e.g. `3D1` = 977, 159 = `9F`. I generated the curves labeled "WDS" by seeking the smallest (or nearly the smallest; I was somewhat unsystematic) b with $|b| \geq 10$ that causes the group order (with $a = -3$ and with that P) to be prime. The secp type-r curves have $a = -3$ with b "generated verifiably at random from a seed using SHA-1 as specified in ANSI X9.62." The secp type-k curves have "an efficiently computable endormorphism" and small $|a|$ and $|b|$. ▲

A few comments about the curves in table 3.3: Some of the standard curves have unappealing prime moduli P, especially in the 128-bit case. This may have been due to the desire of the standards agencies to avoid "patents" on "nice primes." However Daniel J. Bernstein has a web page on which he exhibits prior art to show these "patents" are invalid.

I know of no reason whatever to prefer large random b to the smallest $b \geq 5$. Both are "verifiable," but the smallest $b \geq 5$ is if anything more so – plus it is *much* easier to remember.

Indeed, the only argument I know in favor of verifiably random b is that somebody is worried that there is some magic trapdoor for some kind K of elliptic curves. This kind K has never been noticed in the scientific literature, but we shrink in terror of the possibility that the evil person who generated the elliptic curve might have known about it and generated one of them for poor gullible us to use. However: the "verifiably random" b's employed by the standardization agencies were generated by using secure hash function SHA-1, a $192 \to 160$-bit hash. That means that with at most 2^{160} work (and perhaps a lot less, depending on the properties of K), the forces of evil *could* have generated a random-seeming, but actually type-K, curve.[19] But many of the standard curves are advertised as providing much higher levels of security, e.g. `secp521r1` supposedly is secure against a 2^{256}-operation attack! So I am quite confident that these standard b's are, in fact, nonsense; if they wanted to generate them verifiably at random, they should have done so with a higher-security method.

Now, compare this with my own approach of providing the smallest b. This is immune to any evil design or weakness of SHA-1, and it produces easy-to-remember b. The mathematics of elliptic curves nowhere seems to distinguish between small and large b's so no reason is known why these curves are any weaker than those arising from random b. And finally, the secg standards group *admits* this in that their type-k curves (which they also recommend) have small $|a|$ and $|b|$. (Personally, I think it more likely that the type-k curves will be cracked than mine, so I do not recommend them.[20])

Speed: Bernstein struck again by writing a highly optimized program `nistp224` in 2001 that will perform a random exponentiation within the NIST P-224 curve group in an average of somewhere between 522000 and 1357000 cycles, depending on the processor and certain auxiliary conditions. (This program's runtime and validity is unaffected by the value of b.)

Fundamental open question (Hard rings & fields?). Elliptic curve groups of large prime order are an excellent way to provide somebody with the ability to (1) perform quick group operations in a large finite cyclic group, (2) allow quick conversion of integers to group elements (that is, the integer i is converted to the ith element in the group's cyclic order) but (3) trying to convert in the opposite direction (group elements→integers) is extremely difficult. My question is: is there anything similar if *group* is replaced by *field* or *ring*?

[19] And (what is more likely) it is conceivable that the very design of SHA-1 was chosen in the first place to make that easy. Indeed, SHA-1 *has* recently been "cracked" in the sense that it is possible to produce collisions for it with effort of order 2^{69} hashings rather than the 2^{80} needed for a brute-force attack [164].

[20] Type-k curves have the advantage of allowing faster arithmetic (up to twice as fast) than for random elliptic curves [154].

3.3 Still faster with secret key cryptography

Secret-key cryptography is considerably faster than public key cryptography with the same security level and should therefore be preferred wherever its use is permitted. The greater speed is both because shorter keys may be used, and also because the algorithms (especially the USA's AES=*a*dvanced *e*ncryption *s*tandard [44]) have been designed for high software and hardware speed.

128-bit AES encryption (or decryption) has been implemented to run in about 360 clock cycles on a Pentium-II/200MHz (i.e. 1.8μsec) on average. This is about 4000 times faster than comparable-security public key encryption.

More recent processor chips will encrypt at Gbit/sec speeds.

Even higher speeds are achieveable with custom hardware (although the gain is surprisingly small). Xilinx field programmable gate arrays were devised in 2003 that provide AES encyption at 18 Gbit/sec, i.e. a 128-bit AES encryption may be done with that hardware in only 7.1nsec on average, provided many such 128-bit words are "pipelined." Slower speeds (2Gbit/sec and up) are available in non-pipelined hardware.

4 Algorithmic toolkit

We shall provide brief descriptions of many important cryptographic algorithms: what they do and how they work.

Most of the schemes we describe will be in the Elgamal framework and may be transplanted into the ECC (*e*lliptic *c*urve *c*ryptography) framework by replacing all "$ab \bmod P$" operations where P is a publically known fixed large prime modulus (preferably safeprime), by $a \otimes b$ group operations in a publically known fixed elliptic curve group of (publically known) large prime order. (We shall describe all the more difficult such tranformations explicitly.)

4.1 Fast powering in semigroups

Binary powering: Procedure to compute x^b: Write down the binary representation of the positive integer b, remove the leading 1, and replace each 1 with "SX" and each 0 with "S". So if $b = 13 = 1101_2$ we would get "SX S SX." Input x into a register r. Now read this character-string from left to right. Each time we encounter an S, we *s*quare the register: $r \leftarrow r \otimes r$; while each time we encounter an X we multiply $r \leftarrow r \otimes x$. At the end of this process, $r = x^b$.

If b is N bits long, binary powering performs $\le 2N - 2$ multiplications.

It is easy to see that at least $\lceil \lg b \rceil$, i.e. sometimes at least $N - 1$, multiplications are always needed, so at most a factor of 2 improvement is possible over binary powering. The following algorithm, invented by Alfred Brauer in 1939, achieves that optimal performance in the large-N limit. It performs $N + (1 + o(1))N/\lg N$ multiplications. The fact that no algorithm can do better than this (except for improvements in the "$o(1)$" term) was shown by P.Erdös [57]. We shall assume b is odd since even powers can be handled by doing some squarings after computing an odd power.

Brauer's 2^q-ary powering algorithm to compute x^b:

1. Choose a number $q \approx \lg N - 2 \lg \lg N$.
2. Compute $s = x^2$.
3. Compute and store a table of x^k for $k = 1, 3, 5, 7, \dots, 2^q - 1$ by repeated multiplication by s.
4. Regard b as being written in radix-2^q. Let its "digits" in order from most- to least-significant be b_0, b_1,..., b_r. Compute $z = x^{(b_0)}$ by table lookup.
5. **for** $j = 1$ to r **do** $z \leftarrow z^{(2^q)} x^{(b_j)}$; **end for**. (Here the 2^qth power is done by q consecutive squarings and the b_j power by table lookup of the largest odd factor of b_j, followed by k squarings if b_j contains k factors of 2; by merging these squarings into the other kind they can be made to cost nothing.)

This performs $\le (q + 1)r + 2^{q-1}$ multiplications where $r = \lceil (\lg b)/q \rceil \le \lceil N/q \rceil$.

This method is actually a slight improvement on Brauer's original method. (It also is worth noting that in many fast-multiplication algorithms, squaring is faster than arbitrary multiplication.)

4.2 Fast inversion and square roots in finite groups

Note that $x^{-1} = x^{G-1}$ where G is the order of the group. So if G is known we can perform inversions with the aid of fast powering. (If the group is the multiplicative group of integers modulo a prime P, then $G = P - 1$.)

If $G = 2 \bmod 4$, then we may compute the two square roots $r = \sqrt{x}$ (or prove neither exists) as follows. Iff x is a square then $x^{(G+2)/2} = x$, and then its square roots are $\sqrt{x} = sx^{(G+2)/4}$ where s is either of the two square roots of the identity element 1.

In most of the groups people care about, there is a simpler way to perform inversion than this, but in its absence we can always fall back on this powering method. Furthermore, checking that $x^{G-1} = x^{-1}$ for some random x is an excellent "sanity check" that you both know the correct value of G and have a working powering routine.

4.3 Finding discrete logarithms in "black-box" groups

In some finite cyclic group of known prime order G, suppose we desire to solve $h = g^\ell$ for the discrete log ℓ.

Suppose we have the ability to perform group-operations $a \otimes b$, a^{-1} and a^i where i is any nonnegative integer.

The most obvious (but exceedingly slow) method is simply to try every ℓ in the set $\{0, 1, 2, \dots, G-1\}$. More generally if we originally somehow knew that $a \le \ell \le b$ we could try every ℓ in the integer interval $[a, b]$.

In large elliptic curve groups of prime order, solving discrete logarithm problems is very difficult. However, it is not *that* difficult: it is possible to solve such problems in $O(\sqrt{b - a + 1})$ steps, i.e. roughly square rooting the naive amount of work. Nothing better is known, and it has been argued [146] that no better result is *possible* in "black box" groups. This can easily still be exponentially large, e.g. if G is an n-digit number then $\sqrt{G}$ is an $n/2$-digit number.

Two approaches achieve this: the "baby step giant step" method of D.Shanks, and the "rho method" of J.M.Pollard

[10][129][159]. We may (by pre-multiplying h by g^{-a}) without loss of generality assume that the interval $[a, b]$ is of the form $[0, b]$.

Baby step giant step. Let $r = \lceil\sqrt{b+1}\rceil$. Create an r-entry table of "giant steps," i.e. of the values g^{kr} for $k = 0, 1, 2, \ldots, r-1$. Now for each $j = 0, 1, 2, \ldots r$ compute hg^{-j} (baby steps) and look it up in the table (e.g. by hashing). When a match is found at table entry k, then $\ell = j + rk$.

Pollard rho method. The main problem with the baby/giant method is that it requires enough storage for a table of $\sqrt{b+1}$ group elements. Pollard's method also runs in $O(\sqrt{b+1})$ steps, but its storage needs are tiny. There is a price for that: Pollard's method is randomized and its runtime bound involves a larger constant factor and only pertains to *expected* rather than worst-case runtime.

The idea is to compute the sequence w_0, w_1, w_2,... of group elements where w_0 is chosen randomly and $w_{k+1} = F(w_k)$ for some magic iteration function F. Both F and the initialization procedure for w_0 have to be devised in such a way that the representations of $w_k = g^\alpha h^\beta$ are known to us for each k (that is, all the αs and βs are known). We keep going until, as it inevitably must, a repeat occurs: $w_m = w_n$. Then since

$$g^{\alpha+\ell\beta} = g^\alpha h^\beta = w_m = w_n = g^\gamma h^\delta = g^{\gamma+\ell\delta} \tag{5}$$

with $\alpha, \beta, \gamma, \delta$ known, we may solve $\alpha + \ell\beta = \gamma + \ell\delta$ for $\ell = (\alpha - \gamma)/(\delta - \beta) \bmod G$. (If $\delta = \beta \bmod G$, which occurs extremely rarely, this will not work and we would need to restart to seek a "useful" repeat.)

Pollard's clever low-storage way to find a match is to have *two* walkers through the sequence of w_k (Pollard calls them "kangaroos"), one hopping at speed 1 step per unit time, the other at some slower speed, say 1/2 step per unit time. Because the sequence is ultimately cyclic (shaped like the Greek letter ρ, hence the name "rho method"), the faster kangaroo will eventually lap the slower one so that their locations must eventually coincide[21]. If the function F behaves enough like a random map, so that the walk w_k behaves enough like a random walk, then probability theory shows [83] the expected lengths of both the preperiod and the period will each be $\approx \sqrt{\pi G/8}$.

A suitably random design for the iteration function F is

$$F(w) = \begin{cases} wg & \text{if } H(w) = 1 \\ w^2 & \text{if } H(w) = 2 \\ wh & \text{if } H(w) = 3. \end{cases} \tag{6}$$

where H is some (initially randomly chosen) hash function that maps group elements into the 3-element set $\{1, 2, 3\}$. Defining F this way makes it trivial to deduce the representation $g^\alpha h^\beta$ of $F(w)$ from the similar representation of w.

Pollard's rho-method is parallelizable with linear speedup [162] (although the naive method of parallelizing it yields a much smaller speedup). The trick is to have each processor start from its own random initial w and to post the table of distinguished points (as in footnote 21) on a central bulletin board that all processors can read. As soon as the same distinguished point is generated in two different ways, the job is (usually) done (if not, that processor is restarted at a new random point).

The rho-method runs in $O(\sqrt{G})$ expected steps, and does not take advantage of any knowledge that the discrete log ℓ lies in some short interval $[a, b]$.

Pollard's **lambda method** [129] does take advantage of that knowledge (albeit with some loss of efficiency – it only runs more quickly if $b - a < 0.39G$). There are many variants of the lambda method. We shall just explain one. It involves two kangaroos which start at different places (the "tame" kangaroo at g^b and the "wild" one at $h = g^\ell$ where $0 \le \ell \le b$) and ultimately coincide in location. The iteration function now is

$$F(w) = wg^{H(w)} \tag{7}$$

where H is a hash function that maps group elements to a particular fixed $O(\log(b+1))$-element subset S of the integer interval $[0, b]$. (Pollard recommends the set $S = \{1, 2, 4, 8, 16, 32, \ldots, 2^k\}$ where k is selected so that the average value of S is about $0.5\sqrt{b+1}$.) The tame kangaroo makes $0.7\sqrt{b+1}$ jumps and then stops. The wild kangaroo then starts jumping. If it ever collides with the (now stationary) tame kangaroo, we may solve for ℓ just as in the rho algorithm. If it manages $2.7\sqrt{b+1}$ hops without ever hitting the tame one, then we declare failure (about 25% of attempts lead to failure). After each failure we restart the wild kangaroo from hg^z for some small integer z, continuing until we get a successful run.

It is also possible [162] to parallelize the lambda method with the aid of distinguished points as in footnote 21, and there are many tricks possible here too, but we shall not discuss them. [22]

Quantum computers: All RSA and Elgamal (whether old-style or ECC) cryptosystems would be destroyed if anyone were ever to succeed in building a so-called *quantum computer* because this new kind of computer could solve N-digit integer factoring and discrete logarithm problems in polynomial(N) steps [143]. Personally, I consider this unlikely, and even if it did happen, it would be apparent many years ahead of time that great progress in quantum computers was being made.

4.4 One time pads

"One time pads" are a truly unbreakable cryptographic method. They were invented by Claude Shannon and used for communications between the allies during world war II.

[21] Actually, this 2-kangaroo cycle-detecting method will perform substantially more work than is necessary. A faster idea is to have only one kangaroo, but every time it lands on a *distinguished* point, e.g. one whose hash has first k bits which all happen to be 0, that point is stored. Cycles are detected by performing a table lookup each time the kangaroo lands on a distinguished point. By altering the value of k we can adjust the storage requirements. If we adjust k so that approximately 100 distinguished points are expected to appear during the run, then the runtime will be expected to exceed optimal by $\le 1\%$.

[22] One application of Pollard lambda method is: by using the fact, from Hasse's theorem in footnote 17 and the fact that $x^G = 1$, that x^{P+1} has discrete log base x lying somewhere in the interval $[-2\sqrt{P}, 2\sqrt{P}]$, we may compute that log in $O(P^{1/4})$ steps for some random x, thus determining the order G of an elliptic curve group mod P in $O(P^{1/4})$ steps via a simple algorithm.

The method is this. Alice has a secret message M (a bit string[23]) to send to Bob. She XORs[24] M bitwise with a same-length string R of *random* bits. She sends the resulting randomized message to Bob. Bob then XORs the bits he receives with R, getting M back, thanks to the identity $a \hat{+} b \hat{+} b = a$.

It is necessary for both Alice and Bob to have, before starting, a common random bit string R, known as the *one time pad*. If nobody else knows R, and if R is destroyed immediately after Alice and Bob use it (i.e. it not used again to encrypt some other message!) then all encrypted bit strings are equally likely.

Optical version [115]. Imagine a checkerboard of square pixels (each black or transparent) printed on a translucent sheet. Each 2×2-pixel square subregion may be regarded as a single bit if it is either printed with

$$\text{“0”} = \begin{pmatrix} \blacksquare & \\ & \blacksquare \end{pmatrix} \quad \text{or} \quad \text{“1”} = \begin{pmatrix} & \blacksquare \\ \blacksquare & \end{pmatrix}. \tag{8}$$

If two such sheets are overlaid, then co-located bits which *differ* will appear as a totally black 2×2 square, but if they *agree* then they will appear 50% black, i.e. “grey”:

$$0 \hat{+} 1 = 1 \hat{+} 0 = \begin{pmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{pmatrix}, \quad 0 \hat{+} 0 = \begin{pmatrix} \blacksquare & \\ & \blacksquare \end{pmatrix}, \quad 1 \hat{+} 1 = \begin{pmatrix} & \blacksquare \\ \blacksquare & \end{pmatrix} \tag{9}$$

This provides a visual way – readable immediately without need for a computer – to make 1-time pads. Bob simply overlays the R and $M \hat{+} R$ sheets to read M, which then will spring out in solid black against a grey background. (Or if the second sheet instead were $\overline{R}$, then it would spring out in grey against a black background.)

4.5 Secret key cryptosystems

Alice wishes to send a secret message M to Bob. Both Alice and Bob know (but nobody else does) some random (but fixed) bits K (the “secret key”). Method: Alice repeatedly transforms M by applying one of two specially designed (and publically known) invertible functions F_0 or F_1 to it. She does this k times, once for each of the k bits of K (and using those bits b to determine which F_b to use each time). She then transmits the scrambled message to Bob on an insecure channel. Bob then applies the F_b^{-1} in reverse order to decrypt the message. For sufficiently good designs of F_0 and F_1 and sufficiently long M and K, this sort of scheme is regarded as extremely difficult for any eavesdropper ignorant of K to break (apparently the work required grows like 2^k).

A sufficiently good design of F_0 is this: apply a fixed and publically known random[25] permutation to M's bits, then apply similar fixed random 4096-permutations to each 12-tuple of successive bits in M, regarded as a binary number between 0 and 4095. (F_1 is the same design, but with different fixed and publically known randomness.)

If both M and K are 128 bits long, that seems sufficient to withstand attack by 10^{10} computers each trying 10^{10} keys per second for 10^{10} years; 256 bits should withstand all the computer power that ever will be available on this planet, even running for the age of the universe. The system we have just described is similar to, but in terms of security better than, the AES (USA's “advanced encryption standard” [5]). Note that the software runtime of schemes like this grows proportionally to the *product* of the key and message lengths.

Padding. If the message M is *short* or selected from a small set of possibilities, then it is sometimes possible to break cryptosystems by exhaustive consideration of all possible messages. Therefore, it is recommended to *pad* short messages with a long sequence of random bits appended after the end of the message, and only encrypt padded messages.

Secure hash functions. A secret key cryptosystem $E(\text{key}, \text{message})$ such as AES-128 may be used to produce a “hash function” mapping any bitstring M to a 128-bit “fingerprint.” To do this, let M consist of successive 128-bit blocks m_1, m_2,..., m_ℓ and proceed as follows ([20] schemes 3,5,7):

procedure iterated-hash
- $h \leftarrow \ell$+some constant;
- **for** $i = 1$ to ℓ **do**
  - $h \leftarrow E(h, m_i) \hat{+} h$; $\qquad \triangleright$ Scheme 3 appends $\hat{+} m_i$
- **end for**
- **return** h;

The Europeans have standardized a hash function called WHIRLPOOL which hashes any bitstring to a 512-bit output. (It works almost exactly according to the method we just described, and source code for it is publically available.) Any output size smaller than 512 bits can be got just by using the first n bits of WHIRLPOOL's output.

In 1993 the FIPS standardized a secure hash function called `SHA-1` that will produce a 160-bit-long fingerprint of any bit string of length between 192 and 2^{64} bits; SHA-2/256, SHA-2/384, and SHA-2/512 were similarly standardized in 2003 and have output bit lengths 256, 384, and 512. The SHA-1 (and earlier SHA-0, which has now been fully broken) schemes unfortunately were later found not to be as secure as was hoped [164];[26] the SHA-2 schemes, while still unbroken, are now somewhat suspect.[27]

Verifiably random numbers. By repeatedly feeding the output of a secret key cryptosystem into itself as input, or by encrypting some predictable input stream, we can generate an arbitrarily long stream of “random” numbers [85]. By making the initial input consist of, e.g. the first page of the Bible, it is clear to all that this stream was not generated with some carefully designed malicious goal in mind. (`ANSI X9.62` specifies a particular standard way to generate “verifiably random” numbers from a seed using [the now-broken] `SHA-1`.)

[23] Of course, a “message” can be regarded equally well as a character string, a bit string, or (via binary representation) as a (many-digit) integer.

[24] XOR means the “exclusive or” method of logical combination of two bits. It is the same thing as addition modulo 2: $1 \hat{+} 1 = 0$, $1 \hat{+} 0 = 0 \hat{+} 1 = 1$, $0 \hat{+} 0 = 0$. Equivalently, the XOR of two bits is 1 iff the bits *differ*.

[25] Actually, although random permutations usually would work well, it is preferable to choose them with a certain amount of care to assure high “expansion rate,” fast “mixing,” and generation of the full symmetric group.

[26] “MD5” is another heavily publicized hashing algorithm which has been broken.

[27] Source code for SHA-2/256 is publically available [50].

4.6 Key exchange

Alice and Bob wish to agree on a common secret key (i.e. N-bit binary integer) K which they can later use for the purpose of communicating via a a secret-key cryptosystem, or for some other common purpose.

But when they begin, Alice and Bob have no common information, and all their communications are insecure. So it might seem difficult for them to create a key de novo without somebody else (who is listening to them communicate) also learning it.

Diffie and Hellman's solution is as follows.

1. Alice and Bob agree on a large ($N+1$-bit) prime P and a generator g modulo P (both can be publicized).
2. Alice chooses a random r and Bob a random s, both mod $P-1$ (and they both keep their values private).
3. Alice sends g^r mod P to Bob, while Bob sends g^s mod P to Alice.
4. Alice computes $K = g^{rs} = (g^s)^r$ mod P, while Bob computes $K = g^{rs} = (g^r)^s$ mod P.

Although Eve Eavesdropper learns g, P, and g^r and g^s both mod P, there is no obvious way for her to deduce $K = g^{rs}$ mod P from this, unless she is capable of solving huge discrete logarithm problems.

In the ECC version of this, Alice and Bob publicize a suitable elliptic curve group of prime order and let g be any nonidentity element of it; then all exponentiations are performed within the group and the bitwise representation of x (where elements of the EC group are 2-tuples x, y) is used as the key.

4.7 Public key cryptosystems via RSA one way functions

Bob announces publically the description of a magic "one way function" F_{Bob} that anybody can use to transmit secret messages to him. Alice wishes to send a secret message M to Bob. She computes $F_{\text{Bob}}(M)$ (using the authentic F_{Bob}!) then transmits it to Bob on an insecure channel. Bob then applies F_{Bob}^{-1} to decrypt the message.

This only works if suitable "one way" functions F_{Bob} are easy to find, whose inverse functions both exist, and are difficult for anybody beside the original creator of F_{Bob} to find, even by somebody who fully understands the forward function. Fortunately, all this is true.

The most famous kind of one way function is the "RSA cryptosystem." This works as follows. Let $N = pq$ be the product of two large primes p and q (e.g. each might be 300 digits long). Let e and d be two integers such that $ed - 1$ is a multiple of $(p-1)(q-1)$. (These are easily found by choosing a random 300-digit integer e and then employing the Euclidean GCD algorithm to find the smallest suitable d.) Then the forward encyption function is $F_{\text{Bob}}(M) = M^e$ mod N. The backward, i.e. decryption, function is $F_{\text{Bob}}^{-1}(M) = M^d$ mod N. The reason this works is the Fermat-Euler theorem from number theory [11] which causes $M^{ed} = M$ (mod N). Bob publicizes N and e so that everybody knows what the encryption function $F_{\text{Bob}}(M)$ is, but keeps p, q, and d secret so that only he can decrypt messages. In principle it would be possible to deduce p, q, and d from the published values of N and e, but this is believed to be very computationally *difficult* since essentially the only way anybody knows how to do it is by *factoring* N to find p and q.

4.8 Public key cryptosystems via Elgamal

P is a large publically known prime and M is a nonzero integer modulo P.

I. Massey-Omura. In one Elgamal type scheme (often attributed to Massey and Omura) Alice transmits secret message M to Bob as follows. She computes M^r (mod P) and transmits it to Bob, who then computes $(M^r)^s$ (mod P) and transmits it back to Alice, both using an insecure channel. Here r and s are special nontrivial random integers (mod $P-1$) known only to Alice and Bob respectively, e.g. because they were generated by them randomly shortly before transmission. Alice also knows r^{-1} (mod $P-1$) and Bob also knows s^{-1} (mod $P-1$) since each may compute these quickly by using the Extended GCD algorithm. So Alice now exponentiates with power r^{-1} to compute M^s (mod P), which she transmits to Bob. Finally Bob exponentiates with power s^{-1} to compute M, the decrypted message.

It is believed to be very difficult for anybody ignorant of r and s to deduce M from P and from M^r, M^s, and M^{rs} all mod P. Essentially the only way anybody knows how to do it is by solving discrete log problems to deduce r and s.

The Massey-Omura scheme actually requires no secret keys to exist at all! But it has the disadvantage that if requires a *two-way* conversation between Alice and Bob.

II. Elgamal's own scheme requires only a single (albeit larger) *one-way* transmission. A large prime P, a generator g modulo P, and $h = g^K$ are all made public by Bob, but Bob keeps the large integer K secret. Alice transmits secret message M to Bob as follows:

1. Alice chooses a large integer r at random mod $(P-1)$.
2. Alice computes $z = g^r$ mod P and then $c = Mh^r$ mod P (which is the same as Mg^{Kr} mod P).
3. Alice sends the *Elgamal encryption of* M, namely the 2-tuple (z, c), to Bob.
4. To decrypt, Bob computes $M = (z^K)^{-1}c$ mod P.

Both of these immediately transplant into ECC versions. For example:

II$_E$. Elgamal in elliptic curve group: A large elliptic curve group of prime order G, and a nonidentity element g within it, and $h = g^K$ are all made public by Bob, but Bob keeps the large integer K secret. Alice transmits secret message M to Bob as follows:

1. Alice chooses integer r at random mod G.
2. Alice computes $z = g^r$ and then $c = M \otimes h^r$ (which is the same as $M \otimes g^{Kr}$).
3. Alice sends the *Elgamal encryption of* M, namely the 2-tuple (z, c), to Bob.
4. To decrypt, Bob computes $M = (z^K)^{-1} \otimes c$.

One of the fascinating (and security enhancing) features of these Elgamal systems is that they, if asked to encrypt the same message twice, will generally produce two *different* encryptions because of the included randomness. Hence padding short messages with random bits (as we recommended at the end of §4.5) is unnecessary. However, the price paid for this is a 2:1 expansion of the ciphertext's bit length versus the plaintext. For an Elgamal-type cryptosystem involving arbitrarily small expansion factor see [82].

Multiplicativity (Homomophism): Note that the Elgamal encryption (g^r, Mh^r) of a message M has the property that the elementwise *product* of two such encryptions $(g^r g^s, M_1 M_2 h^r h^s) = (g^{r+s}, M_1 M_2 h^{r+s})$ is the same thing as an Elgamal encryption of the product of the two messages (rgarded as group elements)! This allows multiplication of encrypted quantities without ever decrypting.

Re-encryption: Another nice property of Elgamal encrpytion is that the same message may be *re*encrypted without ever decrypting it (and indeed this may be done by somebody who does not know *how* to decrypt it), e.g. by "multiplying the message by 1" as above.

Cooperative decryption: As we have described it above, Bob, because he knows the magic secret key integer K such that $h = g^K$, is able to decrypt an Elgamal-encrypted message. But suppose "Bob" is really a *set* of m people Bob_1, Bob_2,..., Bob_m, where Bob_j knows an integer K_j such that $K = K_1 K_2 \dots K_m \bmod G$. Then no individual Bob_j is capable by himself of decrypting an Elgamal encrypted message addressed to "Bob," since no Bob_j by himself knows the key K. But all m of the Bob_j's, by working together, can do it. Indeed the crucial decryption step (powering something to the exponent K) may be accomplished by each Bob_j powering it to the exponent K_j, in succession.

It also would be possible for the Bob_j's private exponents to *sum* to K, in which case the product of each's powering could be taken to decrypt the code.

There have also been "threshold" schemes devised in which at least t (for some t with $1 \le t \le m$) of the Bob_j need to cooperate to decrypt a message [48][124][49] . This can be accomplished by having K be the constant term $P(0)$ of a degree-$(t-1)$ polynomial where decryptor j knows $P(j)$ but nobody individually knows $P(0)$. Then the value of $P(0)$ is deducible by Lagrange polynomial interpolation from t values of $P(x)$. Lagrange polynomial interpolation is a weighted sum (the weights L_j are Lagrange interpolation coefficients, and may, after a renormalization, be taken to be public integers); the effect of exponentiation to the power K may be got by private exponentations to the $P(j)$ power followed by public exponentiations to the L_j power and a final producting step to combine it all together. (See also the end of §4.16 for discussion of how to make this idea cheat-proof and see §4.11 for more discussion of the uses of Lagrange interpolation.)

4.9 Digital signatures via RSA or Elgamal

A *signature scheme* allows somebody (call him "Sam") to *sign* a message so that that signature can later be *verified* by anybody else. This verification will prove that the message could only have been signed by Sam, or somebody who knew Sam's private key at the time the message was signed.

I. RSA. One way Sam can sign a message by "RSA decrypting" it using F_{Sam}^{-1}. Anybody can later "RSA encrypt" it using Sam's publicly known one way encryption function F_{Sam}, thus "verifying" that that message could only have come from Sam (or somebody able to run Sam's secret decryption algorithm F_{Sam}^{-1}) and not from any other person.

II. Elgamal's signature scheme (which was the basis of the later "digital signature standard" of the FIPS) is as follows[28]: Sam publishes the large prime modulus P, the generator g, and $h = g^K$ (while keeping K secret) as usual just as in §4.8II. Sam chooses a random nonzero $r \bmod (P-1)$, computes $a = g^r \bmod P$, and computes

$$b = (M - br)H(a, M)^{-1} \mod (P-1), \tag{10}$$

where $H(a, M)$ is a publically known secure hash function[29]. Sam now outputs the tuple $S = (a, b)$ as the signature for the positive integer message M.

To verify this signature, anyone could simply verify that $h^{H(a,M)} a^b = g^M \bmod P$ (since both sides equal $g^{H(a,M)K+rb}$). We would then know that Sam (or anyhow, somebody who knew Sam's secret K value) must have been the signer.

III. Nyberg-Rueppel. A different Elgamal-type signature scheme, due to Nyberg & Rueppel [119], has the advantage (for some purposes) that the signature is not separate from the message. Let $P = 2Q + 1$ and Q be large public primes, i.e. P is a safeprime, and let g be a public nontrivial square mod P. Sam publishes $h = g^K$ while keeping K secret as usual.

To sign the message $M \bmod P$, Sam selects nonzero $z \bmod Q$ at random and computes $r = Mg^z \bmod P$ and $s = Kr + z \bmod Q$. The pair (r, s) is the signature.

At that point anybody may reconstruct the message M, at the same time verifying Sam's signature, via $M = g^{-s} h^r r \bmod P$.

III$_E$. An ECC version of this is as follows. We assume there is a public elliptic curve group of large prime order G. Sam publishes the non-identity group elements g and $h = g^K$ while keeping the integer K secret.

To sign the message M (which now is a group element rather than an integer as in scheme II), Sam selects nonzero $z \bmod G$ at random and computes the group element $r = Mg^z$ and the integer $s = Kr_x + z \bmod G$ (where[30] the integer r_x denotes the x-coordinate of the group element r). The pair (r, s) is the signature.

At that point anybody may reconstruct the message M, at the same time verifying Sam's signature, via $M = g^{-s} h^{(r_x)} r \bmod P$.

4.10 Blind signatures

Suppose Bob writes a secret message on a sheet of paper, puts it in an opaque envelope with a sheet of carbon paper, and hands it to a Notary, who then signs the outside of the envelope and hands it back. The result is that the Notary's

[28] Except that we are presenting an improvement of both because we are making H depend on *both* a and M. See also [72].

[29] Actually EQ 10 is set in the integers and *not* inside the usual multiplicative group mod P. The elliptic curve analogue of this signature scheme (and the FIPS also has a corresponding "elliptic curve digital signature standard") involves using the group-order G everywhere we had written $P-1$; the modulo-P operations become elliptic curve group operations but the mod G operations stay integer operations.

[30] Actually, in place of r_x, any hash function $H(r, h, g)$ could be used.

signature appears on Bob's message, but the Notary never sees the contents of that message.

Blind signatures are the digital equivalent of this story. They involve a conversation between mutually distrustful parties (Bob and the Notary). At the start Bob has his message M; at the end Bob has the Notary's blind signature for M which anybody may then verify in a manner which convinces them that the Notary did sign M. Several protocols have been devised to accomplish this [30][128][3].

The rough plan for obtaining blind signatures has always been this (x and y represent secret key information known only to Bob and the Notary, respectively):

1. (Optional) Notary sends Bob some information I.
2. Bob sends blinded version $B(x, M)$ of his message M to notary.
3. Notary sends signed version $S(y, B(x, M))$ back.
4. Bob unblinds message to get the signed version of his message $S(y, M) = U(S(y, B(x, M)))$.

The question is what magic blinding B, signing S, and unblinding functions U work (if any).

Blinded versions of both the Elgamal DSA and the Nyberg-Rueppel signatures are known [30].

Blinded Nyberg-Rueppel: Let g and $h = g^K$ be public elements of a large public elliptic curve group, of large public prime order G, where K is known only to the Notary (it is the Notary's signing key).

1. Notary chooses random integer γ mod G, computes $I = g^\gamma$, and sends it to Bob.
2. Bob chooses random integers α and β mod G, computes the group element $r = Mg^\alpha I^\beta$ and the integer $B = r_x/\beta$ mod G, and sends B to Notary.
3. Notary computes $S = KB + \gamma$ mod G and sends it to Bob.
4. Bob computes $U = S\beta + \alpha$ mod G.

Then (r, U) is the Notary's Nyberg-Rueppel ECC signature of Bob's original message M, as in §4.9III$_E$ (where, in the notation there, $z = \alpha + \gamma\beta$).

A different blindness scheme, and dated blind signatures. There is a far simpler way to do blind signatures, which for some reason nobody thought of before. It is simply this: do not sign M at all. Instead sign $H(M)$ where H is some publically known secure hash function. Then all "blind" signings can simply be conducted in the open. Furthermore, by signing not $H(M)$, but $H(M)$ *with the date (or other agreed common information) adjoined*, we can get dated signatures (whether blind or not).

Haber & Stornetta [80] imagined a "time stamping service" (TSS). Anyone could send a hash of their document to the TSS, which would send a signed and dated version of that hash back. They pointed out thaat the TSS could be prevented from dishonestly forward-dating the document, by making each signature incorporate bits from that TSS's preceding hash.

4.11 Secret sharing

Alice wishes to transmit a message M to Q different players in such a way that if at least $T + 1$ of those players collaborate, then they can decrypt the message easily, but no subset of $\leq T$ of the players can decrypt the message.

Shamir's scheme [142]. To accomplish this, Alice can regard her message as describing a degree-T polynomial $F(x)$ mod P for some large publically known prime P. (E.g. M is the $(T + 1)$-tuple of its coefficients f_k for $k = 0, \ldots, T$, i.e. where $F(x) = \sum_{k=0}^{T} f_k x^k$.) She transmits the values of the polynomial (mod P) at integer points, i.e. $F(1)$, $F(2)$,..., $F(Q)$, to the respective players via secure channels (or insecure ones, but employing cryptography). Decryption is then just polynomial interpolation mod P.

Note that this scheme is not merely secure under some unproven assumption that some computation is difficult. It is in fact *information theoretically* secure, i.e., the combined information possessed by any T-element subset of the players is insufficient to reconstruct M (no matter how much computing is done) because there are still at least P different possibilities M could have been.

No leak of even partial information. However, as we have described it, a colluding T-element subset of the players could still obtain *partial* information about the secret F, i.e. reducing the number of possibilities. That is fine if P is large enough. However, if desired we may modify this scheme so that not even partial information is obtainable. To do that, make the (now integer) secret be the polynomial's *constant term* $F(0)$. Generate $F(1)$, $F(2)$,..., $F(T)$ at random mod P and then dole out the secret-shares $F(1)$, $F(2)$,..., $F(Q)$. This was, in fact, Shamir's original suggestion and we shall employ it from now on.

Verifiability. Shamir's scheme has the defect that an evil original owner of the secret could have handed out one or more bogus shares, thus not really revealing his secret at all; when the reconstruction later failed, that evil dealer could claim it was not his fault – the problem was instead caused by an evil *player* who faked that share! So that all may recognize evil players and dealers, the protocol may be implemented in the following way.

1. Dealer hands out shares for both his intended secret S, and a random secret R. (Call the shares s_k and r_k respectively, $k = 1, 2, \ldots, Q$. Here s_k is what we previously had been calling $F(k)$ and $S = F(0)$.)
2. For each $k = 1, \ldots, Q$, dealer publically announces $H(r_k, s_k)$ where H is a publically known secure hash function. This *publically commits* him to those s_k and r_k.
3. Each player verifies that her $H(r_k, s_k)$ agrees with what was announced. If not, the player publically complains, and the dealer reveals that r_k and s_k so that the complaint's validity may be judged by all; then all the other players kill whoever lied or refused to follow the protocol.

Later, during secret-reconstruction, the players broadcast their s_k and r_k and players who fail to obey the $H(s_k, r_k)$ condition may be recognized as corrupt. The remaining $\geq T + 1$ of the players reconstruct S and R, recompute the s_k and r_k, and re-verify the $H(s_k, r_k)$. If any of *these* verifications fail, the dealer may be recognized as corrupt. Finally, if the reconstructed S is unsatisfactory, the dealer was corrupt.

Instant Verifiability. Even this "verifiable" scheme still is subject to the objection that a corrupt dealer (handing out s_k that simply do not arise from any S, or supplying an unsatisfactory secret S) would only be detected *later* – it would be better to do so immediately after he doles out the secret-shares. (There is no hope to detect all corrupt *players* at this stage because they may only take the course of evil during the later reconstruction.) Both these objections may be dodged [73][137]

First, the fact that S is a "satisfactory" secret has to be proven via some kind of publically broadcast zero knowledge proof (see §4.13) at the time the shares are doled out. Just what that proof is depends on just what satisfactoryness conditions are to be placed on the secret, so it won't be discussed here.

Second, the dealer must convince everybody that the s_k and r_k he is handing out really *do* come from some S and R, and in particular from the same S that he just proved satisfactory.

To accomplish that, we need three ideas. First, we agree not to use just any old hash function $H(a, b)$, but in fact this one: $H(a, b) = g^a \otimes h^b$ for two public constant nonidentity elements g and h of some public large group (e.g. an elliptic curve group of large prime order).It is convenient to demand that the dealer also publically commit to the value of the secrets themselves (i.e. to the constant term $F(0) = f_0 = s_0$ of his polynomial, and similarly to r_0) by also publishing $H(s_0, r_0) = g^{s_0}h^{r_0}$.

Second, we note [141] that checking any polynomial identity $P_1(x) = P_2(x)$ may be accomplished by trying random x. If the check fails, the identity is false. If it succeeds, then since a polynomial of degree T can have at most T roots, we become convinced of the validity of the identity with confidence $\geq 1 - T/Z$ where Z is the cardinality of the set that x had been randomly sampled from. Also, if more than T distinct X's are successfully tried, then the confidence is 100%.

Third, we note that the reconstruction of a polynomial $F(x) = \sum_{k=0}^{T} f_k x^k$ from its values at $T + 1$ *integer* points x may be done via *Lagrange interpolation* of the form $f_k = \sum_{j=0}^{T} F(x_j)L_{jk}$ where the L_{jk} (which are precomputable if the x_j all are known) are the *rational* Lagrange coefficients. (Specifically, they arise from the coefficients of the polynomial $\prod_{\substack{0\leq j\leq T\\ j\neq k}}(x - x_j)$.) These rationals L_{jk} all become *integers* I_{jk} if they are multiplied by an appropriate pre-computed and public LCM of all possible denominators.

In particular the reconstruction of the value of the secret polynomial's value $F(X)$ at any particular integer location X may be done via $F(X) = \sum_{k=0}^{T}\sum_{j=0}^{T} F(x_j)L_{jk}X^k$. This makes it convenient for us to define $L_j(X) \stackrel{\text{def}}{=} \sum_{k=0}^{T} L_{jk}X^k$ and their integer version

$$I_j(X) \stackrel{\text{def}}{=} \sum_{k=0}^{T} I_{jk}X^k. \tag{11}$$

The key insight of the immediate verification idea of [73] is that, since EQ 11 is just a weighted *sum* with publically precomputable weighting factors $I_j(X)$, its *discrete exponential* $g^F h^R$ may be publically computed, purely from the publically known discrete exponentials $g^s h^r$ of s_k and r_k, by using a *product* of the form

$$(gh)^C \cdot g^{F(X)}h^{R(X)} = \prod_{j=0}^{T}(g^{s_j}h^{r_j})^{I_j(X)}. \tag{12}$$

(where C is some integer constant whose value will not matter).

So: [73] simply suggest *checking* that this same value (at any particular randomly chosen integer X) is got when we instead take the j-product over *any other* (e.g. random) $(T + 1)$-element subset of $\{0, 1, 2, \ldots, n\}$ besides $\{0, 1, 2, \ldots, T\}$. If so, then the s_k and r_k must truly have consistently resulted from genuine degree-T polynomials $F(X)$ and $R(X)$ (or our random selection of X was incredibly unlucky, but we can try again with different random X suggested by outside verifiers, or also just try enough X's to get 100% confidence).[31]

If the check fails, it is publically revealed – immediately – that the dealer was corrupt. If it succeeds, we know the dealer started with a satisfactory secret S [and some other genuine polynomial $R(x)$] and genuinely distributed it to the sharers.

Adding (or subtracting) two shared secrets. Since the sum of two degree-T polynomials is another (and their constant terms sum) sharers of *two* secret integers A, B mod P can *add* them by simply adding their secret-shares. The result is a shared-secret $A+B$. What is interesting here is that the secret sharers can perform certain arithmetic operations ($A \pm B$ mod P) on shared secret integers despite the fact that they do not individually know what those input and output integers are.

No dealer. Self-generation of a random secret by the sharers themselves, is easily possible in this framework by having each player share out its own random secret and then using their sum as *the* secret. In that case, the effect will be just as though a dealer has dealt out a random secret, but now there will be no party who knows it.

Multiplying two shared secrets. It is not as easy to multiply two shared secrets (multiplying the shares would double the degree of the polynomial and hence will not work). Nevertheless, a method for doing so was devised by Gennaro, Rabin, and Rabin [73]. (For an excellently clear discussion of it, see section 3.1 of [86].) Unfortunately, a key step in this method, a zero-knowledge proof that $ab = c$ in appendix B of [73], was later shown [87] to be wrong.[32] Their multiplication method still works if the parties are honest; the flaw is in their attempt to remove the need for that assumption by providing a way to detect dishonesty. There still *are* ways to do the multiplication [18][40] securely, but they are rather grotesque and expensive.

Other tricks with shared secrets. By giving players different numbers of shares (e.g. Alice gets 2 shares, Bob gets 3) we can get the effect of "weighted sharers' where any super-threshold *weighted* set of sharers can reconstruct the secret. By also allowing secret shares themselves to be shared secrets, we can impose hierarchical societal structures on shared secrets – e.g. demand that at least three red sharers plus at

[31] Furthermore this check can be made *non-interactive* by requiring the dealer himself to publish such a check using as X, some hash of all his other publsihed values.

[32] Because it is possible to generate fake proofs which their verifier will accept.

least 5 blue sharers and at least 7 green sharers must cooperate to decrypt the secret. In 1993 the US government tried and failed to impose the "clipper chip" cryptosystem, which was supposed to work like a secret-key cryptosystem but with a "back door" enabling the government (which possessed extra keys) to read encrypted transmissions otherwise only readable by their intended recipient. The clipper chip was based on a secret algorithm depending on secret tamper-resistant hardware. However, a open scheme with similar properties based on public key cryptography can be devised using the secret-sharing methods we just described, where either the intended recipient, or a super-threshold set of (hopefully independent) government agents, could decrypt the message. But even then the whole clipper idea would remain silly because genuine criminals would simply use a cryptosystem without a back door, so that the government could only eavesdrop on people uninterested in being criminals.

4.12 Verifiable shuffles and mixnets

A *verifiable shuffle* is an algorithm A that is given n encrypted messages as input. It then *shuffles* those messages, i.e. permutes them into an apparently random order, while at the same time replacing each encrypted message by a *re-encryption* of that message. Finally, the shuffled and re-encryted messages are output. (We are envisioning Elgamal encryption and re-encryption as in §4.8.)

So far, of course, this is simple and direct to do. What makes it interesting is the further demand that the algorithm also output a *zero knowledge proof* that it has accomplished its task. That is, anybody who examines the input and the output of the algorithm, and the proof, should be able to easily *convince* himself that the output really is a shuffled and re-encrypted version of the input, *but* still should have no idea (more precisely, should have extreme computational difficulty in trying to determine – of course both the decryptions and the shuffle are deducible in principle by trying all possibilities) what the re-encryption transformations were, nor what the shuffle permutation was!

Several authors have devised verifiable shuffle schemes. Here is one particularly simple method:

One way a verifiable shuffler could work. The shuffler, in addition to the input list A and the shuffled and exponentiated output list B, also produces a *third* randomly shuffled list C re-encrypted with random nontrivial exponents, and offers either to prove that $A = C$ or that $B = C$, whichever the verifier prefers. The proof is simply to reveal the permutation and exponents that map A (or B) to C.

If the shuffler is lying, i.e. A and B are not really shuffled and exponentiated versions of each other, then the verifier will detect that lie with probability$\geq 1/2$.

By repeating the procedure s times with a new randomly generated C each time, the probability of an undetected lie drops to $\leq 2^{-s}$.

Non-interactive version. The shuffler

1. Inputs A, outputs B, and also outputs 256 different C's, call them $C^{(1)}$, $C^{(2)}$,..., $C^{(256)}$.
2. Shuffler outputs a standard 256-bit secure hash H of all of the data in A, B, and the $C^{(j)}$.
3. For $j = 1$ to 256: Shuffler demonstrates (as before) that $A = C^{(j)}$ or $B = C^{(j)}$, which one being determined by the jth bit of H.

We do not recommend this algorithm for practical purposes because of the large amount of work involved (due to the large constant 256). We present it merely to make it completely obvious that a non-interactive $O(n)$-step verifiable shuffle is possible.

Less versus more. If, instead of checking that A (or B) is equivalent to C, the verifier only checks a fraction f of the correspondences at random, then the verifier, happily, can do only a fraction f, $0 < f \leq 1$, of the usual amount of computational work (modular exponentiations). She then will gain confidence that the shuffler is not producing anything "too far away" from being a true permutation. For example, if the shuffler had removed 10 entries from the list and inserted 10 fake replacements, then a check with $f = 0.1$ would have had a decent chance of detecting that. This kind of partial checking may well be adequate for many purposes.

More practical version. It is also possible [92] to make this f-scheme almost (or even completely) non-interactive by having the "randomness" actually be (cryptographically strong) pseudo-randomness generated from a small seed "challenge" provided by the verifier. In fact (to be concrete about it) a *single-round*, almost non-interactive verification protocol would be

1. Shuffler outputs all n elements of A, B, and C ($3n$ in all) in encrypted form.
2. Verifier presents random challenge-seed κ.
3. Shuffler uses κ as a pseudo-random seed to generate (in a standard, cryptographically strong way) a 2-coloring of C with $\lfloor n/2 \rfloor$ red and $\lceil n/2 \rceil$ disjoint blue elements. He publishes this coloring. Shuffler now reveals the encryptions used to generate the red C's from the corresponding A's (and reveals the correspondences) and similarly for the B's that come from the blue C's.

After this, the verifier is confident that the shuffler has truly shuffled (and re-encrypted) the n elements of A to get B, except perhaps for a bounded number m (independent of n) of added, deleted, or wrongly-encrypted "cheat" elements (or is confident that the shuffler can break the cryptosystem or enjoyed an exceedingly vast amount of luck). Specifically, if the shuffler produced m cheat-elements, then his chance of being caught would be $\geq 1 - 2^{-m}$.

Objections, and modifications in response. The above scheme can be made completely *non-interactive* by having the shuffler generate κ himself as a standard cryptographic hash function (§4.5) of everything he output in step 1; and provided n is larger than the security parameter (encryption-key bitlength) the scheme should remain secure. However, because an evil shuffler usually can make a probability-ϵ event happen by working $1/\epsilon$ times harder (i.e. trying $1/\epsilon$ different shuffles and proofs, and picking the nastiest one) this completely non-interactive version is probably less desirable in practice than the *almost* non-interactive version we gave.

The scheme we just gave has the disadvantage that it reveals n bits of partial knowledge about the shuffle, namely we know that the $\lfloor n/2 \rfloor$ red items do *not* get shuffled to the $\lceil n/2 \rceil$ blue locations. This in many applications would not matter (e.g. n bits is asymptotically negligible compared to the number $\log_2 n!$ of bits required to specify the shuffle fully). That disadvantage can be eliminated (at the cost of $O(n)$ additional proof and verification work) by zero-knowledge-proving, for each of the red C-items, that it corresponds to either of two A-items, but without saying which (and let all the A-pairs be disjoint). That can be done using the "OR of two ZK-proofs" technique of §4.17.[33]

Another (tiny) disadvantage of this scheme (which again would not matter in many applications) is the fact that it allows a cheating shuffler to have a 2^{-m} chance of cheating on m items without being detected. That chance could be reduced to 2^{-mf} by forcing him to provide f parallel proofs (each with different C and using further bits output by the pseudo-random generator) for any integer $f \geq 1$. Alternatively, that chance could be increased to 2^{-mf} where now $0 < f < 1$, by having the verifier only verify a fraction f (chosen randomly) of the prover's claims about items – saving a huge amount of verification work in the limit $f \to 0+$. This also would save proving-work because the prover would first say (in stage 3) exactly what he was going to prove, then the verifier could say (in stage 4) which proofs he actually wanted to examine (which again could be done using a very short challenge-seed) and then in stage 5 the prover would produce only the requested proofs.

Better (?) shuffles [69][78]. One may make several criticisms of the simple-shuffle scheme described above. First, it is not very efficient if we desire to make the error probability extremely low. Second, it requires *interaction* between the prover and verifier (with multiple seperate interactions and proofs required if there are multiple verifiers who do not trust one another); it would be better if the prover could simply provide *one* proof, which any verifier could then read.

Both these criticisms appeared to be met by Furukawa and Sako's [69] considerably more complicated scheme: which required about $8n$ exponentiation operations for the proof and $10n$ for the verify, with the $18n$ total being reducible "'to the work-equivalent of $\approx 5n$ exponentiations," and which could be converted, by the Fiat-Shamir hash function trick, to only employ non-interactive proofs. But unfortunately, that scheme was later realized by its authors to be flawed because their proofs are not zero-knowledge.

Jens Groth [78] then proposed a different verifiable secret shuffle scheme, requiring only $6n$ exponentiations to construct the proof and $6n$ to verify it. This scheme also appears to be convertable by the Fiat-Shamir hash function trick, to employ only non-interactive proofs. However, I do not understand Groth's method and in view of the historical record of poor correctness of complicated shuffle schemes, I am currently unhappy about trusting it. Therefore in §7.1 we will recommend a simple mixnet scheme reminiscent of Abe's [2] which seems practically useful.

Mixnets: If several mutually distrustful parties consecutively perform verifiable shuffles to n items, then the resulting permutation (assuming at least two of the parties refuse to divulge their random permutations to the others) is unknown to *anybody*. This is called a "mixnet."

4.13 Zero knowledge proof protocols

A zero knowledge proof protocol involves two mutually distrustful parties, the *prover* and the *verifier*. The prover knows some fact and wishes to convince the verifier of it. The goal is to set up a polynomial-time protocol of questions and answers such that, if both obey the protocol, then the verifier ultimately ends up convinced (with probability exponentially close to 1) if and only if the fact is true – if it is false then the verifier will detect a flaw in the prover's argument (with probability exponentially near 1) – *but* during the conversation the verifier learns, essentially, nothing more than the single bit distinguishing "correct" from "flawed."

As a concrete example, suppose Peter wants to convince Vera that he has a 3-coloring of some graph G (whose nodes and arcs are known to both), but *without revealing* what that 3-coloring is.

1. Peter changes the colors randomly (e.g. replacing red by yellow, yellow by blue, blue by red).
2. Peter encrypts the node-colors[34] using a different (randomly chosen) encryption key for each node. He then shows Vera the graph including the encrypted colors for each node.
3. Vera selects a graph-arc A.
4. Peter reveals the decryption keys for A's two endpoints.
5. Vera confirms the decryptions succeed and the two colors of the endpoints are both legitimate and different.

Each time this procedure is performed, Vera's chance of spotting an attempt by Peter to fool her with an invalid 3-coloring, is at least $1/E$ where E is the number of graph edges (assuming Peter does not have enough computational power to, in any reasonable amount of time, dream up fake decryption keys which would miraculously confirm fake equalities). After Ek runs of this procedure without failure, Vera is convinced the graph is 3-colorable with confidence$\geq 1-(1-1/E)^{Ek} > 1-e^{-k}$, but still has no better idea of what that coloring is, than when she started (unless she does enough computational work to break the cryptosystem, which we regard as effectively impossible).

Amazingly, Goldreich, Micali, and Wigderson [77] showed (constructively) that[35]

Theorem 1 (ZKP⊆NP). *If one-way functions exist, then <u>every</u> language in NP has a zero-knowledge proof protocol.*

Examples: Since "shuffling with encrypting" is in NP, the existence of a protocol for performing a verifiable secret shuffle is immediate – although the scheme arising from this general purpose result would be far less efficient than schemes of

[33] If n is odd there would be a leftover A-item, which would have to be dealt with using a triple instead of a pair, just for it.

[34] Since it is silly to encrypt a very short message such as the 3-character string "red," instead we agree to encrypt, say, a string consisting of "redredred...red" with 50 repeats, padded with random characters to make it a 300-letter-long string.

[35] Actually, there are numerous alternative precise mathematical definitions of the concept of "zero knowledge proof," depending on whether the lack of knowledge is protected by assumptions of computational dificulty (and which ones) or information theoretically, whether "probabilitistic proofs" are accepted, etc. We have absolutely no intention of providing those definitions, so in that sense the theorem statements in this paper are subject to criticism.

§4.12 designed specifically for shuffling. "Signature schemes" may also be regarded as just a very special case of zero knowledge proofs, as well as some elements of the verifiable secret sharing scheme of §4.11.

Proof: The proof is actually immediate once we understand the above zero knowledge proof protocol for 3-colorability! Because graph 3-colorability is an NP-*complete* problem [70][155], *any* problem in NP is readily transformed (by a polynomial time transformation procedure known to all) to an equivalent 3-colorability problem, with any solution of the original NP problem being transformed to a 3-coloring and any 3-coloring being back-transformed to a solution. Q.E.D!

Our point is that this general purpose result is a powerhouse.[36] It enables one to design astounding cryptographic algorithms from the "top down": the designer merely says what he wants, and the result magically assures him he can get it, and (if desired) creates it for him! Then only later does the designer need to work on refining everything to make it more efficient.

4.14 (Poor) Efficiency

Unfortunately, theorem 1 provides *inefficient* zero-knowledge proof protocols.

Stockmeyer's reduction [155] will convert any n-variable c-clause 3-satisfiability problem into an equivalent V-vertex E-edge planar graph 3-coloring problem where $V = 3 + 2n + 6c$ and $E = 3 + 3n + 12c$. That means that, in order for the verifier to obtain confidence $\geq 1 - 10^{-s}$ of correctness, the prover needs to perform about $2.3sVE = 2.3s(3 + 2n + 6c)(3 + 3n + 12c) \geq 20 + 13sn^2 + 165c^2$ encryptions (where $2.3 \approx \log_e 10$). In other words, theorem 1 is telling us that any computation involving B *bit*-operations may be turned into a zero knowledge computation with $\leq 10^{-20}$ probability of undetected malfeasance – but now involving about $3312B^2$ *encryption* operations.

Example: Suppose we wish to add and multiply some 32-bit integers: $d = ab + c$. That is about the simplest thing one can do on a modern computer – some do this as a single instruction – and it may be accomplished in somewhere around 1000 bit operations. So according to theorem 1, we can devise a zero knowledge proof protocol to $(1 - 10^{-20})$-convince anybody that we really have numbers a, b, c, d, and we really have computed d correctly – but not revealing what a, b, c, d are – and this protocol will involve a mere 3,312,000,000 encryptions or so. Since 128-bit AES encryption has been implemented to run in about 360 clock cycles on a Pentium-II/200MHz (i.e. 1.8μsec) on average, the prover's part in this protocol would require about 2 hours. With hardware-assisted AES encryption this could come down to as little as 24 seconds. However, keep in mind that the proof-time required grows as the *square* of the number of gate operations in the original computation, so that even with such hardware it quickly becomes infeasible to zero-knowledge-prove any really interesting computation. A computation involving 10^6 instructions instead of just one would require 10^{12} times as much work to zero-knowledge-prove, i.e., 7500 centuries instead of 24 seconds.

So, although theorem 1 assures *for theoretical purposes* the existence of all kinds of wonderful zero knowledge proof protocols with the forward prover-verifier computation requiring "only" polynomial time whereas the effort required to defeat the system is (presumably) exponentially huge – in *practice* these protocols require so much work that this theorem is almost useless.

To get anything practical, we need to either improve the general purpose theorem 1 to make it more efficient, and/or we need to devise a *toolkit* of especially useful and efficient zero knowledge proofs which may be performed far more efficiently than by general methods. We shall do both: the improved theorem 1 in §4.26 and the toolkit starting now.

4.15 Zero knowledge proof that know discrete log

Suppose Peter Prover knows that $g^\ell = h$, for some public non-identity elements g and h in some large public group of known size G, e.g. an elliptic curve group of prime order $G = P$. He wishes to convince Vera Verifier that he knows ℓ, but without revealing the value of ℓ.

Schnorr's protocol:

1. Peter chooses v randomly mod G.
2. Peter computes $t = g^v$ and announces its value to Vera.
3. Vera computes the "challenge" $c = H(g, h, t)$, where H is a secure hash function (arising, e.g. from a secret-key cryptosystem with a key devised randomly by Vera) and announces its value to Peter.
4. Peter computes and announces $r = v - c\ell \bmod G$.
5. Vera may now compute $t' = g^r h^c$ and verify that $t = t'$ and/or that $H(g, h, t') = c$.

Why it works. Obviously, if Peter is an honest prover, Vera's verification will succeed. If Peter is cheating (i.e. does not really know ℓ), then since the hash function is hard to invert, we may assume Peter fixed the value $t' = g^r h^c$ before computing c. Furthermore, Peter had to be prepared for many possible challenges c, since otherwise his probability of successful lying would be too small. But that means Peter could compute many different representations of t' of the form $g^r h^c$ with r and c both known – which implies Peter knew ℓ and hence wasn't cheating!

[36] Some statements *not* in NP can also have zero-knowledge proof protocols. For example graph *non*isomorphism is in coNP but not known to be in NP. If Peter has enormous computational power he can convince Vera that two public graphs A and B are nonisomorphic as follows: Vera chooses one of $\{A, B\}$ at random, chooses a random graph isomorphic to it, and asks Peter which of $\{A, B\}$ (if any) it is isomorphic to. If, every time Vera tries this, Peter always answers correctly, then Vera becomes convinced that A and B must be nonisomorphic (since otherwise Peter could not know which one she had in mind) although she has no idea how Peter is accomplishing what he does.

Later, Shamir [144][145] achieved a complete understanding of just which languages have prover-verifier-interaction zero knowledge proof protocols (under the usual assumption that one-way functions exist). The answer is **Theorem [Shamir]:** *IP=PSPACE*. PSPACE is the set of languages in which membership is confirmable by an algorithm that consumes at most a polynomial amount of memory. Shamir's result shows that membership in any PSPACE language may instead be proven by a zero knowledge *i*nteractive-*p*roof protocol in which the verifier performs only a polynomial amount of work. However (which makes this result less useful for our purposes) the prover may do much more work; again in Shamir's picture the prover is regarded as having infinite computational power.

One-way version: This protocol may also be conducted with only *one-way* communication from Peter to Vera by having *Peter* compute c using a *publically known* secure hash function, in step 3. The original two-way protocol is more secure since Peter has no way of knowing which hash function Vera will employ. But nevertheless, if the public hash function is believed by all to be too hard for Peter (or anybody) to invert and unrelated to discrete logarithms, then this protocol should still be secure.

This trick for converting two-way to one-way (non-interactive) proofs by using the output of a secure hash function instead of random-bit challenges, is often attributed to Fiat and Shamir [61]. It is often employable.

Forgery: Observe that if *Peter*, not Vera, got to pick the challenge c (and was allowed to use any value, not necessarily given by a secure hash function), or if he merely was forewarned, before he started, what the c would be, then Peter would trivially be able to produce a false Schnorr-proof, i.e. even if Peter did not know ℓ. He would simply compute r and t' and announce t' instead of announcing t.

Joint and generalized versions: More generally, Peter may prove that he knows ℓ_1, ℓ_2,..., ℓ_k where $g_1^{\ell_1} g_2^{\ell_2} \dots g_k^{\ell_k} = h$ and g_1, g_2,..., g_k, h are public non-identity elements in some large public EC group. Furthermore, if there are $k \geq 2$ *different* provers P_1, P_2, ... P_k with only the ith prover knowing ℓ_i, then this too may be proven, with nobody revealing their ℓ_i or $g_i^{\ell_i}$ to anybody else (and again this proof may be made non-interactive).

Generalized Schnorr protocol I:

1. For each $i = 1, 2, \dots, k$: Prover$_i$ chooses v_i randomly mod G, computes $t_i = g_i^{v_i}$, and announces t_i to Vera.
2. Vera computes $t = t_1 t_2 \dots t_k$.
3. Vera computes the "challenge"
$$c = H(h, g_1, g_2, \dots, g_k, t_1, t_2, \dots, t_k)$$
and announces it to the provers.
4. For each $i = 1, 2, \dots, k$: Prover$_i$ computes and announces $r_i = v_i - c\ell_i \bmod G$.
5. Vera may now compute $t' = g_1^{r_1} g_2^{r_2} \dots g_k^{r_k} h^c$ and verify that $t = t'$ and/or that $H(h, \dots) = c$.

Note: if one of the provers gets to choose c, rather than Vera, then again he trivially is capable of forging his proof, i.e. getting away with not really knowing his ℓ_i. However, the other provers will still have to know their ℓ_i's in that circumstance, unless the corrupt prover helpfully forewarns them of what c is going to be. (All the forewarned provers can fake their proofs.)

Another variant: suppose there are $k \geq 2$ different provers P_1, P_2, ... P_k with only the ith prover knowing ℓ_i where now the goal os to prove joint knowledge of $\ell = \sum_i \ell_i$ where $g^\ell = h$, with nobody revealing their ℓ_i or $g_i^{\ell_i}$ to anybody else (and again this proof may be made non-interactive).

Generalized Schnorr protocol II:

1. For each $i = 1, 2, \dots, k$: Prover$_i$ chooses v_i randomly mod G, computes $t_i = \prod_{j=1}^{i} g^{v_i}$, and announces t_i to the next-higher prover. Finally $t = t_k = g^{v_1} g^{v_2} \dots g^{v_k}$ is computed and announced to Vera.
2. Vera computes the "challenge" $c = H(h, g, t)$ and announces it to the provers.
3. Provers compute and announce $r = \sum_i r_i$ where $r_i = v_i - c\ell_i \bmod G$. They need to do this in a way that does not reveal the individual ℓ_i's to anybody else, which of course is trivially accomplished by only having them reveal their individual summands r_i to each other.
4. Vera may now compute $t' = g^r h^c$ and verify that $t = t'$ and/or that $H(h, g, t) = c$.

4.16 Zero knowledge test of discrete log equality

Suppose Peter Prover knows that two publically known quantities $x = g^\ell$ and $y = h^\ell$ have the *same* discrete logarithms ℓ (to publically known respective bases g and h) in some group of large prime order P. He wishes to convince Vera Verifier of this – but without revealing what ℓ is.

The procedure (due to D.Chaum & T.P.Pedersen in the early 1990s) is as follows:[37]

1. Peter chooses random r mod G (but keeps it private);
2. Peter computes and prints $a = g^r$ and $b = h^r$;
3. Vera chooses random c mod G and tells it to Peter;
4. Peter computes and prints $z = r + \ell c \bmod G$;
5. Vera verifies that $g^z = ax^c$ and $h^z = by^c$.

after which (assuming the two tests in the final step both succeeded) Vera has very high confidence.

This protocol can be made non-interactive by the usual trick: make c be a standard secure hash function $H(x, g, y, h, a, b)$.

Application to proving Bob encrypted Alice's message: The Schnorr and Chaum-Pedersen protocols may be used to provide a digital version of the following story. Alice supplies a message M to Bob. Bob encrypts it to get E, and hands it back to Alice. Alice needs to be convinced that E really is an encryption of M, but Bob does not wish to reveal the encryption or decryption key.

To do this, Bob could Elgamal-encrypt M as in §4.8 to produce $E = (g^r, h^r M)$. (All operations are in a large publically known cyclic group of publically known order.) Here $h = g^K$ where both r and K are random and known only to Bob, while g and h are public. Alice could now divide $h^r M$ by M to get h^r, and could demand that Bob prove via Schnorr's protocol that he knows r is the discrete log of g^r to base g (or h^r to base h) and that Bob prove by the present protocol that these two discrete logs are equal. That would prove Bob genuinely Elgamal encrypted M.

Extension. Optionally, Alice could also demand that Bob Schnorr-prove he knows Kr, the discrete log of h^r to base g (and hence also that Bob knows K). That would further prove, not only that Bob has *en*crypted Alice's message correctly, but also that he knows the *de*cryption key K.

Application to "plaintext equality test": Suppose we have two ECC Elgamal encryptions $(a, b) = (g^r, M_1 h^r)$ and $(c, d) = (g^s, M_2 h^s)$ and somebody wishes to prove that $M_1 =$

[37] Here g and h are elements of a public EC-group of large known prime order G, while r, c and ℓ are integers.

M_2, i.e. that they both encrypt the same message, but without revealing M_1 or M_2.

In view of the multiplicativity property of Elgamal encryptions (§4.8), this is the same question as deciding whether (ac^{-1}, bd^{-1}) is an Elgamal encryption of 1. There are two ways to answer that:

(i) anybody in possession of the decryption key could provide a zero-knowledge proof of knowledge (§4.15) of the base-g discrete log of ac^{-1} and a zero knowledge proof that it was equal to the base-h discrete log of bd^{-1}. (This solution is unsatisfactory for some purposes because it does not prove $M_1 \neq M_2$, it only proves $M_1 = M_2$.)

(ii) If we exponentiate this to a random power t (nonzero modulo the group order) to get

$$\left([ac^{-1}]^t,\ [bd^{-1}]^t\right) \tag{13}$$

then we still have an Elgamal encryption of 1 (if $M_1 = M_2$) or of $(M_1 M_2^{-1})^t$ (which is a completely random non-identity group element if the group-order is *prime* and if $M_1 \neq M_2$).

Now, anybody in possession of the decryption key (K so that $h = g^K$) may now simply decrypt this Elgamal encryption. The result (1 or a random non-1 group element) provides a zero-knowledge proof that $M_1 = M_2$ or $M_1 \neq M_2$. Furthermore if *nobody* knows K but instead we have a cabal of decryptors each knowing part of K (as discussed in §4.8) then we can get a *distributed* plaintext equality test without any one person ever being capable of decrypting anything.

The "distributed decryption" schemes mentioned at the end of §4.8 can be made immune to cheating by having each partial decryptor provide a ZK-proof (using the method of this section) that he is doing exactly the exponentations he should be doing.

4.17 Zero knowledge proof of one of several discrete logs; ORing and ANDing zero-knowledge proofs

If we know how to prove knowledge of a discrete log ℓ (§4.15) then we trivially know how to prove knowledge of both ℓ *and* another discrete log m. But what if we want to prove knowledge of ℓ *or* m (but perhaps not both)?

So. Suppose Peter Prover knows $h = g^\ell$ or $i = f^m$ (f, g, h, i public) and wants to prove it without revealing either ℓ, m, or the knowledge of which one he knows. Peter now does the following (we assume m is Peter's secret, otherwise proceed with the same protocol with names appropriately swapped):

1. Peter chooses a, b, w randomly mod G and computes $s = h^w g^a$ and $t = f^b$;
2. Peter computes and announces $c = H(f, g, h, i, s, t)$ where H is a secure hash function. (In the two-way version of this protocol s and t would have been publically announced last step and now this challenge c would be computed and announced by the verifier.)
3. $d = w$ and $e = c - d$;
4. $q = a$ mod G and $r = b - ef$;
5. Peter announces the proof (d, e, q, r).

This proof may now be verified by reconstructing $s' = h^d g^q$ and $t' = i^e f^r$ (which if Peter were honest would be the same as s and t respectively) and then checking that $d + e = H(f, g, h, i, s', t')$.

Why it works: Peter is "allowed to forge" one of the two proofs because he can choose its "challenge" d before the commitment c is computed; then the other challenge e is determined.[38] The verifier does not know which challenge was genuine and which forged. This same idea also will work to allow Peter to prove knowledge of one among k discrete logs – he's not saying which – for any fixed $k \geq 2$.

More generally, the same idea will enable producing a zero-knowledge proof of A *or* B whenever we know zero knowledge proof protocols for A and for B alone, and where those individual proofs each depend upon random challenges from the verifier. That is:

$$\text{ZK-proof}_c(A \vee B) \ \equiv\ \text{ZK-proof}_d(A) \wedge \text{ZK-proof}_e(B) \wedge \{d+e=c\}. \tag{14}$$

(The final "+" could instead be a bitwise XOR $\hat{+}$ or a group-multiplication – any of these would yield a valid proof – and the subscripts denote the challenge-values.) If the "random" challenges are in fact replaceable by hash functions (the Fiat-Shamir trick [61] for converting interactive proofs to non-interactive ones) then the *or*ed proof can also be done non-interactively by making the two subproof challenges d and e be linearly dependent as above, i.e. really arising from only one hash value and one forger-chosen value; then all three of $\{c, d, e\}$ are chosen by the prover, with c being a hash of the union of everything the non-interactive version of the subproofs had hashed.

4.18 ZK-proof of at least k-out-of-n statements

Since it has not previously been observed, we point out that the ORing technique of EQ 14 can in fact more generally be used to produce a ZK-proof of k-out-of-n statements, for any fixed integers k,n with $1 \leq k \leq n$. (EQ 14 was merely the special case $k = 1$, $n = 2$ of the following technique.)

We simply demand that the n challenges arising in each of the individual ZK-proofs, obey k nondegenerate linear constraint equations (over a fixed finite field). This allows the prover to "forge" $n - k$ of the proofs by choosing their challenges ahead of time, but the remaining k challenges are then unforgeable since their values then are determined by the k constraint equations – whose right-hand sides were randomly chosen by the verifier.

4.19 Designated-verifier ZK-proofs; deniable signatures

A truly brilliantly simple consequence of the *or*ing trick is the "designated verifier" zero-knowledge proof [93].

Suppose Alice wants to prove some statement S to Bob, in such a way that Bob is convinced, *but* Bob cannot then re-use that proof to convince Carl of S. Alice, in that scenario,

[38] This whole "*or*ing" idea was discovered by both Cramer, Damgård, Schoenmakers, and De Santis, Di Crescenzo, Persiano, Yung independently in 1994.

wants Carl to remain in doubt about whether S is really true, or whether the "proof" is just a forgery created by Bob.

Brilliant idea: *Instead of proving S, Alice will ZK-prove the statement "Either S is true, OR my (i.e. the prover's) name is Bob."*

More precisely the ZK-proof that "my name is Bob" can be a ZK proof of knowledge of Bob's secret key K where Bob had pre-publicized Y with $Y = g^K$ (as in §4.15).

Bob, upon seeing this "S or Bob" proof coming from Alice, will be convinced S must be true, since he knows Alice≠Bob! (Or more precisely, Bob knows that Alice does not know Bob's private key K.) Meanwhile Carl, upon seeing this proof coming from Bob, will have no more reason to believe S than before he saw the proof. (Furthermore, Carl will also be unconvinced of S if the proof comes from anybody who might have gotten or been given Bob's secret key.)

(Another tool for building designated-verifier ZK-proofs are "deniable commitments" as in §4.21.)

Privacy requirement: The prover must be certain that his transmission of his designated-verifier ZK proof to the designated verifier is *private*, i.e. over a *genuinely untapped channel.* (Mere cryptographic protection of the transmission is inadequate.) Otherwise, an eavesdropper could be certain who authored the proof (in the case where encryption was used, the eavesdropper would first require the cooperation of the verifier to perform decryption) and thus could be convinced by it – contrary to the prover's intent.

Another thing to note: The fact that the prover's proof is *only* convincing to one person, in no way prevents that prover from providing some other proof to some other person at some other time.

Deniable signatures: One particular kind of designated-verifier ZK-proof is "deniable signatures." These allow Alice to fulfil her romantic objectives in the following story:

Suppose Alice wants to sign a love letter to Bob in such a way that Bob becomes convinced that it really was Alice that signed it, thus reassuring him that Alice really does love him. *But* Alice does *not* want to allow Bob to be able later to show that signature to some third party Carl and thereby convince him that Alice signed it. Alice, in that scenario, wants Carl to remain in doubt about whether the love letter really originated from Alice, or was just a forgery produced by Bob:

1. Let g be a generator of a large EC group of prime order P. Alice (the signer)'s secret key is K (random integer mod P). Her public key is $Y = g^K$.
2. To sign a message m, Alice computes $h = H(m)$ where H is a publically known secure hash function returning group elements, and returns the signature $s = h^K$.
3. To verify: Alice provides Bob a ZK proof (optionally non-interactive) of $\log_g Y = \log_h s$ (as in §4.16) where $h = H(m)$.

Note: if anybody else, besides Bob, asks Alice for that ZK-proof, she can refuse to provide one or pretend none exist. Therefore, in this sense, Alice's signature is "deniable" – she can only allow the people she chooses to verify it because verification requires her cooperation. However, that is not really satisfactory (especially in the non-interactive case) because Bob can remember the proof, after Alice shows it to him, and then use it to convince Carl. Fortunately for Alice, she can get an even better kind of deniability if she makes this ZK-proof be Bob-only designated-verifier by *or*ing it with a proof of knowledge (§4.15) of Bob's private key. This *or*ing is allowed because both Schnorr's protocol from §4.15 and the proof of discrete log equality from §4.16 that we need in step 3, involve random "challenges."

4.20 Zero knowledge proof of single bit

Let the large prime modulus P, and g and h (both generators mod P with unknown discrete logs to each other as radix) be public.

Suppose Peter Prover chooses r randomly mod $(P-1)$ then computes and publicizes $B = g^r h^{\pm 1} \bmod P$. He wishes to convince Vera Verifier that he really *does* know r, and that the exponent of h really *is* ± 1, i.e. that in some sense B is an encryption of a *single bit*; but Peter does not want to reveal either the value of r or the sign bit. The procedure is as follows:

1. Peter chooses random $s, d, w \bmod (P-1)$.
2. Peter computes and prints $a = g^s (Bh^{\pm 1})^{-d}$ and $b = g^w$ both mod P.
3. If -1, then Peter swaps $a \leftrightarrow b$;
4. Vera chooses random c mod P and tells it to Peter (there is also a one-way communication version of this protocol in which *Peter* computes and announces $c = H(g, h, a, b)$ using a publically known secure hash function H);
5. Peter computes $e = c - d$ and then $t = w + re$ both mod P;
6. If -1 then Peter swaps $d \leftrightarrow e$ and $s \leftrightarrow t$;
7. Peter prints d, e, s, and t;
8. Vera verifies that $e + d = c$, $g^s = a(Bh)^d$, and $g^t = b(Bh^{-1})^e$.

after which (assuming the three tests in the final step all succeeded) Vera has very high confidence.

Why this works I: It is straightforward to confirm that if B really was given by one of the two formulas Peter claimed, then Vera's verifications will succeed. It is also easy to see that Peter does not give away any information that would allow Vera to deduce r (beyond the original information she already had, which would only allow her to deduce it if she could solve huge discrete logarithm problems), nor the sign bit, because everything Peter says is "random." Finally, why does Vera become confident that Peter is not lying?

Why this works II – because it is an ORing: Essentially, Peter is proving his knowledge of the discrete logarithm to the base g of either B/h *or* Bh by *or*ing two Schnorr ZK-proofs (§4.15) using the *or*ing method of §4.17 with the two sub-proof challenge values being d and e. (We apologize for streamlining the proof, which has somewhat obscured this structure.) The reason it is essential that ℓ be *unknown* where $h = g^\ell$, is that we need the *or* to be an *exclusive* or to get a bit, or (equivalently but viewed differently) we need to prevent proof-forgery.

0-1 versus ±1 view of bits: Of course instead of $B = g^r h^{\pm 1}$ we may instead treat $B' = g^r (h^2)^{\{0 \text{ or } 1\}}$ via $B' = Bh$.

Joint version: If there are $k \geq 2$ *different* provers P_1, P_2, ... P_k with only the ith prover knowing r_i, then this too may be proven, with nobody revealing their r_i or g^{r_i} to anybody else, and again this proof may be made non-interactive. The last prover is special since he alone knows the $\pm$ sign and hence the bit; and he wishes to avoid revealing it to anybody else. The proof method is the same – an *or* of two Schnorr proofs (§4.15) – except using the joint/generalized Schnorr protocol II. In this proof, the special prover devises the two subproof challenges (instead of Vera; but doing so in such a way that the two subproof challenges sum to Vera's overall challenge) and informs the other provers what they are. This enables him to forge one of his two Schnorr-proofs. All the other provers act honestly.

4.21 Bit "commitments" and "oblivious transfer"

We just showed how Peter may convince us that we have in our hands, the encrypted version $B = g^r h^b$ of a single bit $b = \pm 1$ – but in such a way that we have no idea what b is. Peter could, in addition, provide us with $H(r, g, h, b)$, where H is a publically known secure hash function. Then *later*, Peter could, if he desires, *reveal* b (by revealing g, h, and the random value r) to us. After checking that $H(r, g, h, b)$ and B really agreed with our previously stored values, we would then be convinced that Peter's bit really was b.

This "bit commitment" scheme is the digital analogue of the following story. We watch as Peter locks one bit (whose value is unknown to us) in a safe. Later, if Peter wants, he can give us the key to the safe.

Although this story may seem silly at first, bit commitment has turned out to be very useful. Trial lawyers will readily understand how that can be: if a witness has previously stated some alleged fact ("committing" himself) then he is unable later to change his story without being revealed as a perjurer. Committing *single bits* is the most flexible and highly controllable form of information committal.

In fact, each round of our general purpose zero knowledge proof of any NP statement (theorem 1 from §4.13) could be viewed just a set of trit-commitments[39] and revelations.

A faster (but less flexible) bit commitment due to Torben Pedersen. Let g and h be public nontrivial elements of a public elliptic curve group of known large public order G. Let $h = g^\ell$ where ℓ is unknown to Peter. Peter may commit a bit b by sending $g^x h^y$ to Vera, where b is the least significant bit of the integer x and where Peter chooses x and y otherwise-randomly mod G. Peter reveals the bit by revealing x and y, whereupon Vera confirms by recomputing $g^x h^y$.

Still faster bit commitment. The elliptic curve version of the bit-commitment scheme we just described is the fastest known bit commitment method for any given (sufficiently high) level of security. However, it is possible to commit bits faster by basing everything on secret-key cryptography instead of public key, which as we remarked in §3.3, is far faster. The price we pay for this speedup is that the prover is now allowed to commit a *non*bit, i.e. random garbage, with that fact only being revealed later when we "open the safe." (In contrast, the above schemes only permit committing genuine bits, with attempts to submit random garbage being rejected immediately.) This is not a tremendous loss, since Peter could simply later refuse to open the safe even in the usual scheme, which would have almost the same effect.

The faster method is this. To commit a bit b, Peter prepares a $2n$-bit string consisting of n copies of b followed by n random bits. He then encrypts it with a secret key cryptosystem. To reveal b later, he reveals the secret key.

Far faster bit committing is possible if Peter is willing to commit n different bits *simultaneously*, and is willing to guarantee that later, when he reveals the bits, he will reveal *all* of them at once. The method is: Peter prepares a $(n + 2s)$-bit string consiting of his n bits, followed by s zero-bits, followed by s random bits. He then encrypts it with a secret key cryptosystem with later revelation by revealing the (s-bit) secret key. Both of these methods work if we assume s is large enough that Peter cannot afford to search $\approx 2^s$ possibilities in an effort to try to produce suitably miraculous fake crypto keys.

Deniable bit commitments [27]:

Bit commitment: Bob commits 0 by sending g^r, and commits 1 by sending αg^r, to Alice, who knows a satisfying $\alpha = g^a$. Nobody else (including Bob) knows a.

Revelation: Bob reveals r to Alice over truly-private channel (not just cryptographically protected, but truly untapped). Alice computes g^r and αg^r to obtain committed bit.

Denial: Although Alice is convinced the bit Bob had committed was indeed the bit that was later revealed, she is unable to convince anybody else (Cindy) of this. If she tries, Bob denies it and argues that Alice could have received $r - a$ instead of r (in which case the bit would have been reversed); there is no way for Cindy to tell which of {Alice, Bob} is the liar.

Oblivious transfer. A different useful primitive operation is "oblivious transfer" [58][100] This is the digital equivalent of the following story: Peter puts bits b_1 and b_2 into two envelopes, labeled 1 and 2. Vera picks one of the envelopes (Peter does not know which) and opens it; the other envelope is burned without opening it.

It is also possible to perform oblivious transfer of multibit *messages*, where Vera gets to read either message 1 or message 2, but not both. This is "1-out-of-2" oblivious transfer. It is also possible to consider "A-out-of-B" oblivious transfer for any $0 < A < B$. In no case is Peter told which message(s) Vera read and which were burned.

While again this story may seem silly, it again has turned out to be very useful in the design of zero knowledge protocols. Oblivious transfer is something like bit commitment, except that Peter cannot be sure which of his bits Vera was able to read. That forces him to be cautious later to avoid being revealed as a perjurer. Meanwhile Vera was unable to read both bits (and in the random-choice version is not sure which bit it was either, and/or Vera does not get to control which bit she reads) preventing her from gaining "too much" knowledge.

Here is Bellare & Micali's protocol [15] for 1-out-of-2 message transfer. We assume there is a large public cyclic group (e.g.

[39] Trits, not bits, since the argument was based on *three*-coloring; true bits could be used if we instead based the proof on 4-coloring.

an elliptic curve group of prime order) of publically known order G, in which a non-identity element g is publicized. There are two parties: the "Sender" and the "Chooser." The sender initially has messages M_0 and M_1, and at the end the Chooser has message M_μ (but no knowledge about $M_{1-\mu}$) where the chooser chooses $\mu \in \{0, 1\}$ and the sender does not know μ.

1. Sender picks a random non-identity group element C and announces it to Chooser.
2. Chooser chooses a random nontrivial r mod G, computes $K_\mu = g^r$ and $K_{1-\mu} = C/g^r$, and sends K_0 to Sender.
3. Sender computes $K_1 = C/K_0$, chooses random s_0, s_1 mod G, computes $e_0 = g^{[s_0]}$, $e_1 = g^{[s_1]}$, $E_0 = H(K_0^s)\hat{+}M_0$, and $E_1 = H(K_1^t)\hat{+}M_1$, then sends (e_0, E_0, e_1, E_1) to Chooser, where H is a publically known secure hash function and $\hat{+}$ denotes bitwise XORing.
4. Chooser computes message $M_\mu = H((g^{[e_\mu]})^r)\hat{+}E_\mu = H(K_\mu^{[s_\mu]})\hat{+}E_\mu$.

Note: in this protocol Sender, if he wanted to be uncooperative, could have swapped M_0 and M_1 before using them. In that case, the effect would be as if μ were being chosen randomly and would be unknown to *both* parties.

Oblivious transfer is more powerful than bit commitment. It is possible to perform bit commitment by using oblivious transfer (but obviously the reverse is not possible). Simply transmit n bits b_1, b_2,..., b_n whose XOR is the desired bit b (but they are otherwise random) via 2-to-1 oblivious transfer in such a way that the choices are (1) the correct b_k and (2) a random bit. Then, tell the chooser (after he has chosen) which of (1) and (2) were which, so that he knows (but the Sender does not) which of the bits he received were correct. Then, to reveal b later, retransmit all the b_k in the clear. (Warning: There is some probability that this scheme will fail by either accidentally revealing b or by allowing the Sender to make an undetected false revelation later. But these probabilities are exponentially small as a function of n as n is made large.)

Message commitment may also be done via oblivious transfer, for example by simply doing one bit commitment for each bit of the message. However, there is a much faster way which often is acceptable: Peter can send a long message M to Vera by 1-out-of-2 oblivious transfer of each of the bits of $F_K(M)$ (with the other choice being a random bit), where $F_K(M)$ is a secret-key encryption of M with publically known key K Then Peter reveals to Vera which of the choices was genuine in each case, so that Vera then knows (although Peter does not) which bits of $F_K(M)$ she successfully received. Peter later can reveal M.

Oblivious transfer is necessary and sufficient for two-party equality testing. Suppose Alice has a number X and Bob has a number Y, and they wish to test whether $X = Y$, but do not wish to reveal any information about X or Y (aside from the fact that $X = Y$ or $X \neq Y$) to each other or anybody else.

This problem is surprisingly difficult. Numerous unsatisfactory solutions were considered in [59].[40]

One idea is for Alice and Bob to encrypt their numbers and compare the encryptions, but if the numbers are known a priori to come from a small set of possibilities, then that would be defeated by exhaustive search. Another idea is for Alice and Bob to announce the encrypted numbers to Carl and ask him whether they were identical. But Carl might lie.

When we say that oblivious transfer is *necessary* to solve this problem, we mean, more precisely, that if the $X = Y$ test were possible in which only Alice learned the test-result, it could be used to implement one form of oblivious trit-transfer. Specifically, let Alice and Bob pick a sequence of random numbers from the set $\{1, 2, 3\}$ and repeatedly perform equality tests. If $X = Y$, then Alice knows Bob's number. Otherwise, she knows Bob's number is one of two possibilities, i.e. she "fails to receive" Bob's number. Oblivious transfer as originally defined by Kilian [100] involved bits being transmitted Bob→Alice and either being received (in which case Alice knew she'd received them and knew their value) or not (in which case Alice knows she failed to received the bit) with Bob not knowing which bits were received and which were not, and with the question of which being determined by randomness, not by either party.

Here is how [59] do the equality test with the aid of oblivious transfer. (The procedure will incur some probability 2^{-s} of error; we assume X and Y are both n-bit numbers.)

1. Choose s large enough that the error probability 2^{-s} is acceptably small.
2. Alice chooses n pairs (A_{k0}, A_{k1}) of random s-bit numbers, $k = 1, 2, \ldots n$.
3. Bob chooses n pairs (B_{k0}, B_{k1}) of random s-bit numbers, $k = 1, 2, \ldots n$.
4. Alice computes $T_A = \sum_{k=1}^{n} A_{kx_k}$ where the $\sum$ denotes a bitwise XOR and where x_k is the kth bit of Alice's number X.
5. Bob computes $T_B = \sum_{k=1}^{n} B_{ky_k}$ where y_k is the kth bit of Bob's number Y.
6. Alice chooses one number $B_{k\sigma}$ from each of Bob's pairs by 1-out-of-2 oblivious transfer. She of course chooses the correct $\sigma = x_k$ each time.
7. Bob chooses one number $A_{k\sigma}$ from each of Alice's pairs by 1-out-of-2 oblivious transfer. He of course chooses the correct $\sigma = y_k$ each time.
8. Alice computes the XOR-sum of her new numbers and XORs it with T_A to get S_A.
9. Bob computes the XOR-sum of his new numbers and XORs it with T_B to get S_B.
10. Alice and Bob reveal S_A and S_B, and then $X = Y$ if and only if $S_A = S_B$.

This procedure is subject to the criticism that Alice and Bob might not want to cooperate with each other. In particu-

[40] A humorous one is: hash the numbers into the set of valid telephone numbers. Alice phones her's.
GUY ON OTHER END OF PHONE: Hello?
ALICE: Hi. This is Alice. I'd like to leave a message for Bob.
GUY: Huh? Who in hell are Alice and Bob? You must have the wrong n-
ALICE: ⟨click⟩.
Then Bob phones his number. BOB: Hello, this is Bob. Did Alice leave me a message?

lar Alice might cooperate up until the final step, and then lie about the value of S_A. In that case Bob would (almost certainly) conclude $X \neq Y$ while Alice would know the truth.

To defeat that criticism, the revelations could proceed bit by bit in order, terminating as soon as a disagreement was seen. (See also §4.24.) That would still permit one party to lie or cease cooperating at some point, but that person could only have 1 bit more information than the other, i.e. at best half the error probability of the other, which is a far smaller advantage.[41]

A different (but very similar) solution to this $X = Y$ problem was given much later by [26], who seemed unaware of [59]. That solution also has a 1-bit asymmetry. The authors of [26] then remarked that the question of designing an efficient similar protocol to settle the "millionaire's problem[42]" of deciding whether $X > Y$, remains unsolved. We shall solve it in §4.27.

4.22 Zero knowledge proof number is in interval

We can use the single-bit proof of §4.20 to prove that an encrypted *number* z is k bits long, i.e. to prove that $z \in \{0, 1, 2, \dots, 2^k - 1\}$.

Specifically, let the large prime modulus P, and let g and h (both random fixed generators mod P) be public.

Suppose Peter Prover chooses r randomly mod $(P-1)$ then computes and publicizes $B = g^r h^z \bmod P$. He wishes to convince Vera Verifier that he really *does* know r, and that the exponent z of h really *is* k bits long. But Peter does not want to reveal the values of r or z.

The procedure: Let the binary representation of z be $z = \sum_{j=0}^{\ell-1} z_j$ with $z_j \in \{0, 2^j\}$. Peter now simply sends Vera the values of $B_j = g^{r_j} h^{z_j} \bmod P$ for each $j = 0, 1, 2, \dots, \ell-1$ and uses appropriate 1-bit proofs (albeit based on h^{2^j} rather than h, and on $\{0, 1\}$ rather than $\{\pm 1\}$ bits, see §4.20) to convince Vera that each B_j encodes a single bit. Here Peter chooses the r_j randomly subject to the constraint that $\sum_{j=0}^{\ell-1} r_j = r$. Finally, Vera checks that $B = \prod_{j=0}^{\ell-1} B_j \bmod P$.

It is also possible to prove (without revealing ℓ) that a discrete logarithm ℓ lies in any particular public finite set, *not* required to be a contiguous interval, using the method of §4.17. This requires a number of modular exponentiations growing linearly with the cardinality of the set.

Extension to arbitrary interval sizes. By proving that $x = x_1 + x_2 + \dots + x_\ell$ where x_i is proven to be in $[0, 2^i - 1]$, (where only the subscripts that correspond to the locations of the 1-bits in the binary expansion of B are used) we can prove in zero knowledge that x lies in the interval $[0, B-1]$.

4.22.1 A flawed procedure by Fabrice Boudot

Fabrice Boudot was unhappy with the fact that the first procedure only permits intervals of cardinality 2^ℓ (the extension to arbitrary interval sizes seems onerous) and also was unhappy with the fact that the work (as measured, e.g. by the number of modular exponentiations required) grows proportionally to ℓ for large ℓ; he would prefer slower growth.[43] Boudot, in a seemingly excellent paper [25], was able to devise a more complicated zero knowledge proof of membership in any *arbitrary* integer interval $z \in [a, b]$, and in which the number of modular exponentations ultimately apparently grows only like $O(1 + \log \ell)$ when $\ell = \lg(b - a + 1)$ is large.[44]

But unfortunately, Boudot's "improvement" is not an improvement! That is because Boudot depends for his security *both* on the hardness of discrete logarithms (in elliptic curve groups, if desired) *and* on the hardness of factoring large composite integers. Therefore, really, to get the same security as our ECC-based scheme involving $O(\ell)$ modular exponentiations each with N-bit numbers, Boudot must use modular exponentiations with $\approx N^3$-bit numbers (to be more precise see footnotes 13 and 15). So when N is large, just *one* of Boudot's exponentiations requires more runtime than our entire "worse" procedure even if $\ell \approx N$.

4.22.2 A new procedure that repairs Boudot's flaws

It would be better to alter Boudot's techniques so that they (1) depend only on the hardness of discrete logarithms (in elliptic curve groups, if desired) and *not* on the hardness of factoring, (2) thus then represent a *true* speedup.

We shall now show how to accomplish that.

Boudot's work involved 3 main ingredients: (a) the $O(\ell)$-step zero knowledge proof described above that some integer is in the interval $[0, 2^\ell - 1]$; (b) a zero knowledge proof that a number is a square; (c) a zero knowledge proof that a number that the prover knows is in some interval, is in some (significantly larger) interval. (This is proving something weaker than what the prover knows, but this weakness allows the proof to be done fast.) Boudot's plan, roughly, was to express x as the sum of a square s^2 and a small number (of size $O(\sqrt{x})$), prove the small number small-enough using (c), prove s^2 square using (b), and prove s small using (c) or a recursive proof.

A simpler plan would involve ingredients (b) and (c) only. It is to prove membership of x in an interval $L \le x \le U$ by proving the nonnegativity of $x - L$ and $U - x$. To prove nonnegativity we can use Lagrange's theorem that any number is the sum of 4 squares (for a fast algorithm to find 4 such squares see [132]). But Groth [79] saved work with the more clever suggestion[45] of proving $4y + 1$ is the sum of *three* squares to prove an integer y nonnegative.

Boudot's ingredients (b) and (c) both depended on the hardness of integer factoring and thus Boudot's methods remain of no real interest. We now show how to replace both ingredients by with new ZK-proofs *not* dependent on the hardness of integer factoring and only dependent on the hardness of the discrete logarithm problem in elliptic curve groups.

[41] Note also that Alice could consistently lie about X and act as though it were X'. This would cause her to learn whether $X' = Y$, which would be useless information for Bob but somewhat useful for her. Also, either Alice or Bob coul simply cease cooperating at any point and then neither would learn anything. These problems are inescapable.

[42] Alice and Bob are both millionaires and wish to determine who is richer, but without revealing their wealth.

[43] And of course, the runtime growth of the more general procedure that can handle membership in any finite set, is far worse still.

[44] Actually, this estimate oversimplifies the situation, i.e. is not strictly true.

[45] Any nonnegative integer of the form $4y + 1$ is a sum of three squares.

ZK-proof for a sum of squares. Given that $a^2+b^2+c^2=4y+1$, how do we ZK-prove this? I suggest we prove

$$(T^a)^a(T^b)^b(T^c)^c = (T^y)^4T^1 \tag{15}$$

where T is a generator in an elliptic-curve group. More precisely, one should instead use a homomorphic ECC-cryptosystem because T^x does not actually hide x if x is a small integer, but the ElGamal-like cryptosystem $x \to (g^r, h^r m^x)$ where g, h, m are public constant EC elements with $h = g^\ell$ and ℓ is the secret key, does hide x from any body ignorant of the secret random integer r. Equation 15 above gives our idea in a simplified notation; the real idea is to use the full tuple $T = (g^r, h^r m^x)$ instead of T^x everywhere T^x (for $x \in \{a, b, c, y, 1\}$) occurs in that equation, where tuple exponentation and multiplication is elementwise.

A ZK-proof may now be produced by the usual tools for proofs of discrete log equalities, etc.; we omit details.

ZK-proof of smallness. We also need to ZK-prove that a, b, c are small enough so that $|a|, |b|, |c| \ll \sqrt{P}$ where P is the (enormous) known cardinality of the elliptic curve group (which in applications is going to be far larger than y). That assures there is no modular "wraparound" when computing $a^2+b^2+c^2$ so that then y must be nonnegative. The smallness proof may be accomplished with the aid of this new

ZK-proof of membership in a much-widened interval. Let b and t be public positive integers and let g and h be independent random fixed nonidentity public group elements in some large public elliptic curve group of large known public prime order G. Peter Prover knows an integer x and knows $0 \le x \le b$ for some public integer constant b. He wishes to convince Vera Verifier of the weaker statement that $|x| \le B \stackrel{\text{def}}{=} 2^t b$ but does not wish to reveal x.

1. Peter chooses random r mod G.
2. Peter computes and announces $F = g^x h^r$ mod N (thus publically "committing" to x).
3. Peter: **repeat**
   (a) Pick random w with $0 \le w \le B-1$ and random n mod G.
   (b) Peter computes $W = g^w h^n$ and announces it to Vera.
   (c) Peter asks Vera for a random secret t-bit integer c, and Vera obligingly generates and supplies one.
   (d) Peter computes $D = w + xc$ and $E = n + rc$.

   until $cb \le D \le B-1$.
4. Peter computes $i = h^E$.
5. Peter sends (D, i) to Vera.
6. Vera verifies $cb \le D \le B-1$ and $W = g^D i F^{-c}$.
7. Peter proves knowledge of the discrete log (which is E) of i to the base h, but without revealing E, by using Schnorr's protocol from §4.15.

Why it works. Vera reasons that Peter had to be prepared to receive almost any t-bit random challenge number c. Admittedly, there are a few c's which Peter will not handle, in which case he performs another iteration of the repeat-until loop. But if Peter performs too many iterations, Vera will become suspicious: the probability that any particular iteration of the loop will fail (i.e. require another go) is $\le 2^{-t}$. (If we add to the protocol, as we really should, that Vera refuses to believe Peter if he employs more than S loop iterations, then there is an acceptably small probability$\approx 2^{-St}$ that the whole protocol will fail even if Peter is honest.) Note that $W = g^D i F^{-c} = g^{D-xc} h^{E-rc} = g^w h^n$. If $|x|$ had been too large, namely if x had disobeyed Peter's claim $|x| \le B \stackrel{\text{def}}{=} 2^t b$, then there usually would not have existed any D with $cb \le D \le B-1$ with $D - xc = w$ where we know that Peter had to choose w (since he publically committed to it via announcing W) *before* knowing what c was going to be. So Vera is fairly confident that Peter's claim must be correct (or else Peter simply got very lucky – the probability he was the recipient of such luck is $\le 2^{-t}$ – but by repeating the proof R times the probability of such luck decreases to 2^{-Rt}).

Why does Vera have zero knowledge about x? It is not hard to see that D is precisely uniformly randomly distributed within its allowed range $cb \le D \le B-1$ and hence there is no information available from D alone. If Vera knew both D and E she perhaps could then use the 2 linear equations $D = w + xc$ and $E = n + rc$ to constrain the 4 unknowns w, x, n, r and thus gain partial knowledge. However, she does not know E. With only one equation she gains no knowledge about x or w because r and n are completely random.

Finding sums of squares: Finally: how hard is it to find 3 squares that sum to a given number (which is something our prover must do)? Rabin and Shallit [132] gave a fast algorithm.

Groth [79] noted that if the number $x = 4y+1$ is small enough so that comparable-size numbers can easily be factored into primes (this is often true in applications) then the following simple method works:

1. Choose a random even integer a with $a^2 \le x$.
2. Factor $x - a^2$ into primes.
3. If not all the primes are of the form $4m+1$, then go back to step 1.
4. Each prime p of the form $4m+1$ in the factorization has a representation as a sum of two squares $p = A^2 + B^2$, and the representation is unique up to the signs of A and B. Find it using Cornacchia's algorithm ([37] section 1.5.2) and randomize the signs.
5. If $p = A^2+B^2$ and $q = C^2+D^2$ then $pq = (AC+BD)^2 + (AD-BC)^2$ can be used to build up a 2-square representation of the prime-product $x - a^2 = b^2 + c^2$. We are now done: $x = a^2 + b^2 + c^2$ is a random representation of x as a sum of three squares.

Heuristic arguments[46] indicate that the loop in steps 1-3 will iterate $O(\log x)$ times in expectation. Hence this procedure should run quickly.

[46] The "probability" that a number N has all prime factors of form $4m+1$ is, assuming each prime factor "decides" to be of form $4m+1$ or not by flipping a coin, about 2^{-F} where F is the number of distinct prime factors of N. But $2^F \le d(N)$ where $d(N)$ is the number of divisors of N. And a theorem of Dirichlet says that $d(N) = \ln N + O(1)$ on average. So the rough "probability" random N has all prime factors of form $4m+1$ should be $\ge 1/\ln N$.

4.23 Proof of El Gamal encryption of a single bit and of a number in an interval

The previous "proof of knowledge of a single bit," "proof of knowledge of one of two discrete logarithms," and "proof of equality of two discrete logs" will not be quite good enough by themselves for our later needs for homomorphism-based voting.

What we shall need for that purpose is a proof of one of two *Elgamal* encryptions. That is, Peter Prover knows the 2-tuple

$$(g^{\alpha}, h^{\alpha} i^{\pm 1}) \tag{16}$$

(and in fact knows the value of every letter g, h, i, α in it) which is an Elgamal encryption (§4.8) of the message $i^{\pm 1}$. Here g, h, i are public group elements in a public elliptic curve group of large prime order G and α is a random integer mod G. Note this is the Elgamal encryption of a *single bit* (the $\pm$ sign).

Peter wishes to prove to Vera that this 2-tuple (s, t) is the Elgamal encryption of either i or i^{-1} but without revealing which, and without revealing the random (mod G) integer α.

This comes down to proving that $\log_g s = \log_h(t/i)$ *or* $= \log_h(ti)$. We've already seen in §4.16 how to prove the equality of two discrete logs without revealing either, and we've already seen how to prove knowledge of one OR the other among two discrete logs – it is just that we now need to do both at the same time.

Here is the proof protocol [42]:

1. Peter announces the 2-tuple $(s, t) = (g^{\alpha}, h^{\alpha} i^{\pm 1})$.
2. Peter chooses w, r, d at random mod G;
3. Peter computes $a_1 \leftarrow g^r s^d$; $b_1 \leftarrow h^r (t i^{\pm 1})^d$;
4. Peter computes $a_2 \leftarrow g^w$; $b_2 \leftarrow h^w$;
5. If -1 then Peter swaps $(a_1, b_1) \leftrightarrow (a_2, b_2)$;
6. Peter sends (a_1, b_1, a_2, b_2) to Vera;
7. Vera selects a challenge integer c mod G at random and sends it to Peter;
8. Peter computes $e \leftarrow c - d$; $q \leftarrow w - \alpha e$;
9. If -1 then Peter swaps $(d, r) \leftrightarrow (e, q)$;
10. Peter sends (d, e, r, q) to Vera;
11. Vera verifies $c = d + e$, $a_1 = g^r s^d$, $b_1 = h^r (ti)^d$, $a_2 = g^q s^e$, and $b_2 = h^q (y/i)^e$.

If Peter instead wished to prove $(g^{\alpha}, h^{\alpha} i^{v})$ was an Elgamal encryption of an exponentiated number v lying in the interval $[0, 2^{\ell} - 1]$ then he could now proceed similarly to §4.22 by proving that this 2-tuple to be the elementwise product of the Elgamal encryptions of ℓ single bits based on i, i^2, i^4,..., i^{ℓ} but using $\{0, 1\}$ rather than ± 1 bits.

4.24 Co-signing, dating, and "bit-by-bit release" technique

Cosigning problem. Suppose Alice and Bob both have a document M they both want to sign, *but* neither can countenance the horrible possibility that the other could get ahold of a copy of that document signed only by themself and *not* by that other (which would allow them to misuse that once-signed document later).

The solution is for Alice and Bob to each compute their signatures and then release them to one another one bit at a time in alternation. Each bit comes accompanied by a zero knowledge proof of its correctness. That way, if at any point one of them ceases to cooperate, then they are at best "1 bit ahead" in the race to compute the remaining unrevealed bits by exhaustive search (and hence can expect at best a factor-2 speedup).[47]

We can provide a concrete implementation of this idea using the Nyberg-Rueppel signature scheme III_E of §4.9. Alice begins by releasing r in her signature (r, s) in its entirety. Since r is just a random group element this reveals nothing by itself. Then, to release s one bit at a time, she proceeds as follows.

1. Alice publishes g^s (and/or g^{-s}) thus publically "committing" to s and also allowing Bob to verify (by checking that $M = g^{-s} h^{(r_x)} r$) that she indeed has a valid signature in mind. Suppose s is N bits long.
2. Alice releases the most significant bit b of s. She proves this bit valid by showing that g^t (where $t = s - B$ where $B = 2^{N-1} b$, i.e. where $g^t = g^s / g^B$) represents the discrete exponential of an $(N-1)$-bit number t. She does this using the interval-containment zero-knowledge proof for t given in §4.22.
3. Alice can now proceed similarly for the next most significant bit (only using t and $N-2$ instead of s and $N-1$), and so on.

By reusing the single-bit proofs inside the procedure of §4.22 this runs in $O(1+N)$ modular exponentiations total (without reuse it would have been $O(N^2)$).

More precise explanation of bit-by-bit-revelation: Let g and h be random (but fixed) public nonidentity elements in a large public elliptic curve group of known large prime order G. Here is how Alice may reveal an ℓ-bit secret integer s bit-by-bit to Bob.

1. Alice chooses a random r mod G (but keeps it secret)[48].
2. Alice begins by revealing $B = g^s h^r$. (This publically commits her to s.)
3. Let the binary representation of s be $s = \sum_{j=0}^{\ell-1} z_j$ with $z_j \in \{0, 2^j\}$. Alice sends Bob the values of $B_j = g^{r_j} h^{z_j}$ mod P for each $j = 0, 1, 2, \ldots, \ell - 1$ and uses appropriate 1-bit proofs (albeit based on h^{2^j} rather than h, and on $\{0, 1\}$ rather than $\{\pm 1\}$ bits) to convince Bob that each B_j encodes a single bit. Here Alice chooses the r_j randomly subject to the constraint that $\sum_{j=0}^{\ell-1} r_j = r$.
4. Alice proves each of the B_j encodes a single bit using the protocol of §4.20 but for $\{0, 2^j\}$ bits rather than $\{0, 1\}$ or $\{\pm 1\}$ bits.
5. Bob checks that $B = \prod_{j=0}^{\ell-1} B_j$ mod P. He is then convinced that the B_j encode each of the ℓ bits of s.
6. Alice also provides $H(r_j, g, h, z_j)$ for $j = 0, 1, \ldots, \ell - 1$ where H is a publically known secure hash function.

[47] This also allows Alice and Bob to *date* the document they are signing, by simply including the date – which they must agree on – as part of the document before they begin.

[48] This is not the same r as in the above signature scheme. We here are disregarding any applications to signatures or anything else, and just considering here how to reveal s bit-by-bit.

7. Now Alice may reveal these bits one by one. To reveal B_j as in §4.21, she simply announces z_j and r_j. Bob may then verify the hash values and that Bg^{-z^j} is the product of the other B_j's.

This requires Alice to compute $2+4\ell$ modular exponentiations while Bob computes 4ℓ.

Blindness. This all also works for blind signatures using the blindness-by-hashing idea at the end of §4.10.

4.25 Other zero knowledge proofs

Here is a zero knowledge proof that Peter Prover knows the prime factorization of some composite number N.

1. Vera chooses a random x mod N and sends x^2 mod N to Peter.
2. Peter uses his knowledge of N's prme factorization to compute a random square root r (mod N) of x^2.
3. Peter chooses a random y mod N.
4. Peter computes $s = y^2$ mod N and announces it to Vera.
5. Peter sends a secret-key encryption of y and a secret-key encryption of yr to Vera (using randomly chosen secret keys).
6. Vera chooses which one she wants revealed.
7. Peter reveals either y or yr by sending Vera the appropriate one of the secret keys.
8. Vera decrypts and checks that $y^2 = s$ or that $(yr)^2 = sx^2$.

After t repetitions of this protocol (with different randomness each time) Vera has confidence $1-2^{-t}$ that Peter really knows the factorization of N. This works because it is known that the ability to find square roots mod N is equivalent to being able to factor N. It is zero knowledge because Peter does not reveal $r = \sqrt{x^2}$, but only a random residue or random multiple of r.

A zero knowledge proof that a given number N is a product $N = pq$ of two safeprimes, is presented in [31]. This is useful for convincing somebody that an RSA modulus is safe to use, but without revealing the decryption key. (This reference also shows how to ZK-prove a number is a "pseudoprime.")

Damgård and Fujisaki [45] advanced a scheme in which integers i may be "committed" in the form $g^i h^r$ mod N where N is a fixed public product of two large unknown safeprimes, and efficient zero knowledge schemes are given for checking that $a = b + c$ or that $a = bc$ where a, b, c are committed integers. For methods for performing finite field arithmetic in zero knowledge, see [39]. Some of these ideas were used in a voting scheme [46]. Lipmaa [108] then proposed zero knowledge proofs for arbitrary diophantine equation statements. We shall not discuss all these proofs. They all are less desirable than ours because they depend on the hardness of factoring and hence are 50 times slower (cf. §3.2) than same-security proofs which depend only on the hardness of the discrete logarithm problem in elliptic curve groups.[49]

Although these references make it possible to do finite field and exact-integer ring operations in zero knowledge (and see §4.27 for multiparty finite field ring-operations) reasonably efficiently, *real* arithmetic using *inexact* representations of reals (such as IEEE-754 arithmetic) is a complete nightmare. It is precisely this problem which presently prevents the better multiwinner voting schemes from being implemented in a secure and efficient way.

Fundamental open question (approximate arithmetic): Can you invent an efficient scheme for performing some reasonable kind of inexact real arithmetic (such as IEEE-754) in zero knowledge or via secure multiparty computation?

4.26 Faster general purpose zero knowledge proofs

Theorem 2 (NP⊆linear time zero knowledge). *An operation of any publically known forward-flow logical circuit with W "wires" (including inputs and outputs) and G "logic gates" (each either a 2-input NAND or 1-input NOT gates) may be verified in zero knowledge (i.e. without revealing the boolean logical states of either its input or output bits (or of any internal bits) by a procedure involving $\le 1W + 6G$ (prover) plus $\le 8G$ (verifier) modular exponentiations or (to get more security with less work and fewer stored bits) exponentiations in an elliptic curve group of prime order. If desired, the communication in this protocol can be made entirely "one way" (from the prover to the verifier).*

Proof: The Prover encrypts each bit $b = \{0 \text{ or } 1\}$ on each "wire" inside the circuit as $g^r h^b$ where r is randomly chosen and secret (and different for each wire), and prints out a picture of the circuit diagram, with each wire labeled with the encrypted value of its logical-state bit. Each such encryption may be proven legitimate via the method of §4.20.

The fact that a NOT gate works may be proven as follows. Suppose its input bit b has encryption $e = g^r h^b$, and its output bit $1 - b$ has encryption $f = g^s h^{1-b}$; then the product $m = ef^2$ is $m = g^{r+2s} h^{2-b}$. The Prover now proves that the exponent $2-b$ of h inside the formula for m lies in the interval $[1, 2]$ – or equivalently that the exponent $1 - b$ of h inside the formula for m/h lies in the interval $[0, 1]$. This may be done using the protocol of §4.20.

The fact that a NAND gate works may be proven as follows. Suppose its input bits b, c have encryptions $x = g^r h^b$ and $y = g^s h^c$, and its output bit d has encryption $z = g^t h^d$; then the product $m = bc^2d^4$ is $m = g^{r+2s+4t} h^{b+2c+4d}$. The Prover now proves that the exponent $b + 2c + 4d$ of h inside the formula for m lies in the interval $[1, 4]$, i.e. in the set $dcb = \{001, 010, 011, 100\}$ of binary representations. (Or equivalently that the exponent $b + 2c + 4d - 1$ of h inside the formula for m/h lies in $[0, 3]$.) This is a cardinality-2^ℓ interval-membership proof and may be performed using the protocol of §4.22. Q.E.D.

Remark. The prover sometimes finds it desirable to *reveal* the output bit (e.g. to prove it is 1, to prove "satisfiability"). Such bit-revelations are possible as described in §4.20.

This scheme is new. It is a substantial speed improvement over theorem 1: Using `Zmodexp` to perform 4.66msec 512-bit modular exponentiations, a 1000-gate circuit would be verified

[49] One interesting application of these was Lipmaa et al. [109]'s proposed cryptographic scheme for performing Vickrey auctions [163]. We shall discuss our own solution to that in §4.27.

in ≤ 8000 modular exponentiations, plus a smaller number of secure hash function computations. This would require about 40 seconds on Pentium-II/350MHz (as opposed to 2 hours as in §4.14). Even more importantly, the runtime now grows only *linearly* (not quadratically) with the number of gates in the circuit being simulated; hence a 10^6-times larger circuit could be handled in 1.3 Pentium-years.

Despite this improvement, this must still be regarded as very slow, so slow that this technique must be employed only a few times (if at all) during any election, and certainly not, e.g. once per vote.

The next theorem will show how to go faster by using secret key cryptography in place of public key, and by allowing a some exponentially small probability 2^{-s} of error or undetected malfeasance and by allowing many rounds of interaction. By combining ideas of Kilian, Micali & Ostrovsky [101] with those of Brassard, Chaum, Crepeau [28][50] we get the fastest method known, and it is also quite simple. It is highly recommended.

Theorem 3 (Fast ZKP for NP). *Any operation of a G-gate publically known flow-forward logical circuit (all nontrivial kinds of 2-input gates are permitted) may be proven in zero knowledge, i.e. without revealing knowledge of any input or intermediate bit. (The output bit may be revealed, or not, whichever the prover wishes.) The ZK-proof consists of (1) prover commits two strings of $12G$ bits each, (2) verifier transmits information to prover, (3) prover replies, (4) one of the two committed bit-strings (chosen by the Verifier) is revealed. The whole 4-step proof is then repeated s times, after which the probability that the verifier has failed to spot a flaw in any non-proof is $\leq (3/4)^s$. If the bit-string committals are performed using AES-like secret key cryptography with a K-bit key (as in §4.21) then the entire proof & verify procedure requires $O(GsK)$ work.*

Proof: The **first lemma** [101] is that any ZK proof which is of the following "subset revealing" form:

1. Prover sends verifier some bit commitments;
2. Prover and verifier communicate;
3. Verifier selects a subset S of the bit committments;
4. Prover reveals those bits;
5. Prover and verifier communicate some more.

may be converted to an equivalent "two-envelope" ZK proof of the following form:

1. Prover sends verifier *two* bit-string commitments;
2. Prover and verifier communicate;
3. Verifier selects one of the two bit strings and verifier reveals it in its entirety;
4. Prover and verifier communicate some more.

The only price paid is that the probability that the Verifier will successfully reject a flawed proof, may *halve.* However, by repeating the whole proof $O(s)$ times, the chance of undetected bogusness is reduced to $\leq 2^{-s}$.

The **proof** of this lemma is astonishingly simple. The prover's two bitstrings are $M\hat{+}R$ and R, where M were the bits committed in the original proof, and R is a random bitstring of the same length (and $\hat{+}$ denotes bitwise XOR). When, in the original proof, the verifier asks that a subset S of the committed bits be revealed – in the new proof, the prover replies with the alleged values of those bits of M and of the corresponding bits of R. If the prover is lying about any bit, then the corresponding bit in either R or $M\hat{+}R$ must disagree with the Prover's claim. The verifier now chooses which of these two strings he wants revealed. After its revelation, the probability the prover's lie is revealed by a disagreement is $\geq 50\%$. In other words, if the prover had to lie, the original proof would have revealed that lie, but the new proof will only reveal it with $\geq 50\%$ probability. Hence, as claimed, the probability that the Verifier will successfully reject a flawed proof, at worst halves. Q.E.D.$_1$

The **second lemma** [28] is that there is a subset-revealing ZK proof of operation of any G-gate, W-wire flow-forward logical circuit with 2-input gates. The proof works as follows. The Prover first computes the bit-values present on every wire in his circuit. He then randomly independently "scrambles" the truth tables defining each of his logic gates by randomly taking one of the $24 = 4!$ possible permutations of its 4 rows, and randomly choosing to complement each of its 3 columns (total of $24 \cdot 2^3 = 192$ possible scrambled forms).

a	b	$c = \overline{a \wedge b}$	a'	b'	c'
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	0	0
1	1	0	0	1	1

Figure 4.1. The truth table defining a NAND gate (on left); and a scrambled form of it, got by swapping the last two rows and then complementing the first two columns (on right). ▲

The scramblings must be consistent: all truth table columns corresponding to the same wire in the circuit must all be complemented or all left uncomplemented. This is achieved by randomly and independently choosing to complement (or not) each wire. (The final output wires, however, are never complemented, at least those whose logic states the prover wishes to reveal.) The new circuit is then equivalent to the old one.

After producing this circuit, the Prover *commits* to it by committing all bits in all truth tables and all bits on all wires.

The Verifier now *challenges* the prover to either

1. Reveal every committed bit in every truth table.
2. Reveal only the one row in each truth table that actually got *used.* (This also, automatically, reveals the states of all input and output wires to all gates.)

If the challenge is successfully met, then the prover either knows that a genuinely-equivalent circuit was described, or that its operation worked – and yielded the claimed output (if the prover is claiming a certain output value). It is impossible for both to be true unless the circuit actually *does* work with some input to yield the claimed output. Therefore the verifier would spot a bogus proof with probability$\geq 1/2$.

[50] Nobody seems to have pointed out the fact that these two methods may trivially be combined. The Kilian et al. idea [101] is brilliantly simple but seems insufficiently well known, which is perhaps because it only appeared in two conferences, not a journal, and only in an abbreviated form containing many minor errors and self-contradictions (although readers might not have *realized* the minorness of those errors).

This proof is zero knowledge since the scramblings make all wire logical values equally likely, i.e. the revelations give no information whatever away. Q.E.D.$_2$

Now by combining these two lemmas we get a zero knowledge proof in which two $12G$-bit strings are committed and in which any attempt to provide a bogus proof would be detected with probability$\geq 1/4$. Each bitstring may be committed via an AES encryption of $36G$ bits as described in §4.21 and one may be later revealed by an AES decryption (with a key provided by the Prover).

By redoing this proof s times consecutively, the probability of an undetected flaw becomes $\leq (3/4)^s$. The total amount of work is $O(GKs)$ assuming we are using AES with a K-bit secret key. Q.E.D.

This is a far faster protocol. Consider again the case of a 1000-gate circuit. To get probability$< 10^{-20}$ of any undetected flaw in a proof, we need to perform 160 rounds. Each round requires two AES encryptions of 36000-bit strings and then one decryption. The total time required for all that on a Pentium-II/200MHz using 128-bit AES is 1/4 second, i.e. 160 times faster than the preceding theorem. If a much larger probability (≈ 0.1) of having an undetected flaw in the proof were regarded as acceptable, then an additional factor-20 speedup would be attainable (total proof time now 1/80 second).

Zero knowledge simulations of fairly large computations now become possible, e.g. a sequence of 10^6 computer instructions, each equivalent to 1000 gate operations, would be ZK-provable with probability≈ 0.1 of having an undetected flaw in the proof, in 3.5 hours on the Pentium-II/200MHz. Thus in a sense theorem 3 shows that any sequence of computer instructions may be simulated in zero knowledge (with confidence 0.9) with a slowdown factor of about 2.5 million.

Interaction. The protocol in theorem 3 is still subject to the criticism that it involves many rounds of *interaction* between the prover and verifier. Kilian and Petrank [102] overcame that criticism by showing how to make a *non*interactive ZK proof for a G-gate circuit consisting of $O(G \log G)$ committed bits, with probability of undetected flaw in the proof being $\leq G^{-c}$ for some positive constant c. This is the same speed as theorem 3 if $\log G \leq O(s)$, but now without interaction. Unfortunately the constants hidden in Kilian and Petrank's O and c are incredibly enormous and hence that accomplishment is presently of purely theoretical interest.[51]

4.27 Secure general multiparty computation

Suppose there are some number $Q \geq 2$ of mutually distrustful parties. Their goal is to compute the output of an *arbitrary* publically known forward-flow circuit made of G logic gates (possibly including "randomizing gates") from its boolean inputs, and to do so in such a way that

1. Both the input bits, the output bits, and all intermediate logical states are "shared secrets" never known by any of the parties individually, and indeed never knowable by any ($\leq T$)-element subset of the parties acting collusively, and
2. External observers viewing the communications among the parties can become convinced that they are performing the computation correctly, (even though these observers also will be unable to know any of these bits),
3. The total computational work is equivalent to some polynomial(G, Q) number of exponentiations in some large public group (preferably small), and
4. The total number of bits communicated is equivalent to some polynomial(G, Q) number of group elements (preferably small), and
5. Preferably, the parties share out the work and communication roughly equally.
6. Nevertheless any subset of $> T$ of the parties, by cooperating, may easily deduce and reveal any desired subset of these bits, in particular the output bit.

This is called secure general multiparty computation (SGMPC).

Theorem 4 (Secure verifiable multiparty computation). *Secure general multiparty computation is possible.*

Similarly to the way the original proof of theorem 1 could be viewed (as we remarked in §4.21) as simply some bit-commitments and revelations, it is possible to do arbitrary secure multiparty computations instead with the aid of oblivious transfer [100]. However, that is not the way we are going to prove it. We are going to discuss several proofs which achieve better and better performance (smaller polynomial bounds).

First proof (sketch): secret sharing. The input bits are each shared out, initially, using Shamir's integer secret sharing scheme from §4.11 (use the "instant verification" version and with the "integers" in the present case being just 1 or 0). As we remarked there, it is possible to *add*, subtract, or multiply two shared-secret integers, so that we may perform logical ANDing via multiplication, and logical NOT by subtracting: $y = 1 - x$.

Caveat: when I first wrote this proof, I was under the impression that [73] was a valid paper; that would have led to a bound of order $O(Q^3 G)$ on the number of exponentiations and the number of communicated group-elements. But in fact, as we mentioned in §4.11, their secure multiplication primitive is insecure [87]. Although it is known to be possible [18][76][40] to repair this error and create a genuine secure multiplication primitive, the ways I am aware of are baroque and have large complexities. That makes this proof far less algorithmically attractive than I had originally thought. Q.E.D.

Second proof (sketch): Beaver's multiplication trick

Donald Beaver in 1991 invented a brilliantly simple idea for avoiding the difficulty of multiplying shared secrets. In a preparatory phase, he first generates and distributes a pool of triples (a, b, c) of random constants with $c = ab$ to all secret sharers.

Whenever, in the proof above, the protocol calls for multiplying two shared secrets F and G to get FG (also in shared-secret form), we proceed as follows.

[51] Kilian told me "we could have improved them from 'incredibly enormous' to merely 'awful,' but it would have taken too much space in the journal." He also added re theorem 1 that "using PCPs there are ways to make graphs such that either you can 3-color it or you have to miscolor a constant fraction of the edges."

1. Parties agree on a particular as-yet-unused shared triple (a, b, c).
2. All compute $d = a + F$ and $e = b + G$.
3. The parties un-share the resulting shared secrets d and e, to get their (random) decrypted values.
4. The sharers compute $eF - db + c$. Note: (i) this is algebraically the same thing as FG. (ii) because e and d are publically known integers, the sharers can do these "multiplications" by doublings and addings only, i.e. using the linearity property that adding two shared secrets yields the shared version of the sum of the secrets. (This dodges the problem that multiplying two shared secrets does not yield the shared version of the product of the secrets.)

We have omitted some details, such as how to generate and zero-knowledge prove (in a distributed fashion) the validity of the initial triples. (E.g. see section 3.2.4.2 of [86] and section 3.3 of [88].) This scheme may be implemented to require $O(GQ^2)$ exponentiations and group-element transmittals, plus $O(Q^3)$ in the preparatory phase [88].

Third proof (and the one leading to the fastest algorithm): "mix and match" [91] The idea of this proof is inspired by the proof of theorem 3, the fastest method known for proving general NP statements in zero knowledge.

First, just as in theorem 3, the parties produce a random equivalent circuit by randomly independently "scrambling" the truth tables defining each of the logic gates by randomly taking one of the $24 = 4!$ possible permutations of its 4 rows, much as in table 4.1. However, the particular permutations are chosen by use of a *mixnet* and hence are not known to any party individually. After this is done, the circuit is described by a set of "blinded truth tables" describing which encrypted-input-bit combinations to each logic gate result in which encrypted-output-bits.

Second, we need to *match* the input-bit of one logic gate with the output-bit of its predecessor gate, or (if there is no predecessor-gate) with the initial (Elgamal encrypted) input-bits to the circuit – although all these bits are Elgamal encrypted with different randomness. These matches are accomplished by distributed "plaintext equality tests" (§4.16) performed in gate forward-flow order. Finally, the last gate produces the output bit in encrypted form; the players jointly decrypt it. The joint decryptions need to be accompanied by ZK-proofs by each player that they are correctly doing their part in each; those are readily provided by using the techniques of §4.15 and §4.16.

The whole mix and match protocol requires $O(QG)$ exponentiations to produce a verification of circuit operation on one input-bit word, assuming nobody cheats. If anybody cheats, the cheater is immediately spotted and the procedure is terminated. (It can be restarted with the corrupt party excluded.) Q.E.D.

Warning about security models [87][88]: Several security models have been proposed in the SGMPC literature, with the most conservative one assuming at most T out of the Q parties may cheat at any stage of the protocol, with the identities of the cheater-set possibly continually changing, and the protocol still grinds through to successful completion while guaranteeing *information-theoretic* (rather than merely computational) security under the assumption that untappable private communication channels are available between any pair of parties, in addition to a broadcast channel. But we here have only addressed the simplest model (which seems sufficient for our voting purposes), in which there are corrupt and honest parties, and it suffices to publically expose the corrupt ones if and when they cheat, and to prevent (too few) corrupt parties from accomplishing anything nasty, and everything depends on computational complexity assumptions such as the difficulty of solving discrete logarithm problems and cracking cryptosystems, and the only available communication method is broadcasting.

Fault tolerance: It is also possible to pre-transform the circuit, using standard methods, into a equivalent (but larger) fault-tolerant circuit, and then simulate *it.* That can give extra immunity against malfeasance.

Millionaire's problem: Alice and Bob know two numbers X and Y respectively and want to know whether $X < Y$ but do not wish to reveal any information about their numbers. In 2001 it was claimed [26] that this problem still lacked any efficient and fair solution, where by "efficient" we mean "with work polynomially bounded in terms of the number of bits in X and Y" and by "fair" we mean that neither Alice nor Bob can gain a significant knowledge-advantage over the other even if by cheating.

We now provide such a solution. In fact, we show how Alice and Bob can compute *any* function $f(X, Y)$ in an amount of work polynomial in the number of bit-operations that would normally have been required to compute $f(X, Y)$ if there were no zero-knowledge requirements.

1. Alice takes the bits of X and shares each of them out among herself and Bob using [73] using the "instant verifiable" secret sharing scheme there with "no partial knowledge" of our §4.11 and [73].
2. Bob takes the bits of Y and similarly shares each of them out among himself and Alice. Note these protocols involve commitments to X and Y and are based on the hardness of inverting secure hash function and of solving discrete logarithm problems (optionally: in elliptic curve groups). So neither Alice nor Bob can lie without detection[52] *once they commit* to X and Y.
3. Now using the procedures of §4.11 and [73] to perform ab (AND) and $1 - a$ (NOT) operations on shared-secret bits, we simulate a comparator circuit for deciding if $X > Y$; eventually ending up with the shared-secret bit B saying whether $X > Y$ (or more generally, $B = f(X, Y)$). This is exactly as in the proof of theorem 4 using $T + 1 = Q = 2$.
4. Finally, Alice and Bob "unshare" B i.e. combine their partial information about B to deduce B. (They have no motivation to unshare any other shared information they have, hence they will not release any knowledge about X and Y aside from that inherent in the value

[52] Nothing stops Alice from lying all along about X though, in which case she learns something about Y but reveals nothing about the true X. Also at any point either party could cease to cooperate in which case the protocol would shut down. If any party tried to lie after the initial committal stage they would be detected by the other, who would then stop cooperating. These limitations are inescapable. Compare footnote 41.

of B.) This reconstruction has to be done with a little care: if Bob simply reveals his secret share, Alice could then lie about hers, learning the truth while Bob learns garbage. So therefore, Alice and Bob each reveal 1 bit at a time of their secret-shares. Each bit is accompanied by a zero knowledge proof that bit is accurate. (Since in §4.11 all the shares are publically committed using Elgamal, proving bits of them may be accomplished with the aid of the techniques of §4.22 for proving membership of discrete logs in integer intervals.) See the "bit-by-bit-release" technique of §4.24. If anybody quits cooperating or lies, then the other immediately terminates their role in the protocol and is only "1 bit behind" in the task of trying to find out B by exhaustive search of the remaining bits. That search is therefore only a factor of two larger.

5. Finally with the shared secret reconstructed, they both know B and the job is done.

The runtime is $O(N+K+1)$ exponentiations, where N is the number of bit-operations to compute $f(X,Y)$ – in the case of the millionaire's problem we may just take N as the number of bits in the numbers representing the millionaire's wealths – and K is a security parameter (bit width of the numbers involved in modular exponentations)[53]

Although it might seem at first that the "millionaire's problem" is a joke problem of no real-world importance, that impression is incorrect. Here is a very similar important real world problem: **auctions.**

In a *Vickrey auction* to buy some object, the winner is the bidder whose bid is highest, but he only pays the *price* bid by the *second*-highest bidder. [More generally, if there are to be $w \geq 1$ co-winners, then they each pay the $(w+1)$th-highest bid price.] The brilliant point of this kind of auction is that it has the property that bidders are motivated to provide *honest* assessments of their valuation of the object as their bids [163]. But that is only true if those bids are *secret* and if the seller is *honest*. (If some bidder were to know what the highest competing bid was, he then could take advantage of the knowledge; a cheating seller could also announce a false value of the second highest bid to get more money, or collude with a bidder to provide an artificially high second-highest bid.)

The cryptographic question then is: how to provide that secrecy and prevent that dishonesty while still allowing the auction to proceed. Specifically, we want all bidders to submit bids, but for *nobody* to know anything about what any of those bids are (complete privacy for maximum prevention of bid-rigging), with the sole exception that the value of the second-highest bid and the name of the highest bidder are announced. Even if bidders *want* to reveal their bids, they should be unable to do so in any convincing manner. And finally, we want verifiability – all are convinced these announcements were correct.

This is achieveable as follows. All players secret-share each bit of their (binary integer) bids among the other bidders. They run a multiparty protocol to compute the output of an appropriate circuit.[54] They then un-share this output to find the result of the auction. As long as the majority of the bidders are not corrupt, all privacy is protected. Note: each secret share, in the sharing scheme of §4.11, are random numbers providing no information about the bid, and the accompanying secure-hash values provide no aid either – they merely proevent sharers from lying about the share they received, or secret-distributors from lying about the shares they distributed.

[53] Also, Alice and Bob can communicate encrypted in their final stage in which case they can learn f but external eavesdroppers cannot.

[54] To find the highest bidder, the circuit makes a single-elimination tournament between the bidders, with only the higher bidder in each pairoff continuing on to the next round; and to find also the second highest bidder we make also a "consolation tournament" among the losers. Each pairoff is accomplished via a binary comparison circuit. At the end we can make the circuit computes the identity of the top bidder (but discard his bid) and the second-top bidder's bid (but discard his name).

Modular multiplication (mul): $O(N \log N)$ work [138][19].
Modular exponentiation (modexp): $\leq \min\{2N-2,\ N+(1+o(1))N/\lg N\}$ muls.
Inversion x^{-1}: ≤ 1 modexp in general groups; trivial in EC groups.
Find discrete-log ℓ so $g^\ell = h$, given $0 \leq \ell \leq b$: $O(\sqrt{b})$ muls.
Secret key crypto (AES): KN work for either encrypt or decrypt.
RSA public key crypto: encrypt=decrypt=1 modexp.
Elgamal public key crypto: encrypt=decrypt=2 modexps.
RSA digital signature: sign=verify=1 modexp.
Elgamal or Nyberg-Rueppel digital signature: sign=1, verify=2 modexps.
Blinded Nyberg-R signature: sign=3, verify=2 modexps.
Our simpler blind signature: same as plain signature.
Schnorr ZKP that know discrete log: proof=verify=1 modexp.
Neff [116] ZK verifiable shuffle of n numbers: shuffle&prove= $8n+4$, verify= $12n+4$ modexps.
Groth [78] ZK verifiable shuffle of n numbers: prove≈ $6n$, verify≈ $6n$ modexps.
Our ZK verifiable shuffle of n numbers: shuffle&prove= $(1+s)n$, verify= sn modexps, prob$\leq 2^{-s}$ of proof's undetected falsity.
ZKP that $\log_a b = \log_c d$: prove=verify=2 modexps.
ZKP have a single bit: prove=3, verify=4, create-the-bit=1 modexp.
Bit-commitment: commit=1 modexp by the committer, reveal=1 modexp by the receiver.
Fast n-bit string-commitment: commit=1 AES(of $\max\{3n, K\}$ bits) by committer, reveal=1 AES by receiver.
2-message-to-1 oblivious transfer: sender=3, chooser=2 modexps.
ZKP Bob has Elgamal encrypted Alice's message (§4.16): prove=verify=3 modexp. (And encrypt=2).
ZKP $0 \leq x < 2^\ell$: prove$\leq 3\ell$, verify$\leq 4\ell$, x-creation=1 modexp.
ZKP that $|x| < 2^t b$ (if $0 \leq x < b$ with t, b public positive numbers): $O(s)$ modexps, 2^{-ts} probability of failure.
ZKP of Elgamal encryption of single bit: initial encrypt=2, proof=5, verify=8 modexps.
ZKP of Elgaml encryption of i^x with $0 \leq x < 2^\ell$: prove$\leq 5\ell$, verify$\leq 8\ell$, initial encrypt=3 modexp.
Bit-by-bit revelation of N-bit secret: Revealer= $2+4N$ modexps, Watcher= $4N$ modexps.
ZKP for any G-gate W-wire NAND/NOT logical circuit: prove$\leq 1W+6G$, verify$\leq 8G$ modexps.
Fast ZKP for any G-gate logical circuit: prove= 2AES(of $\max\{36G, K\}$ bits), verify=1 AES, prob$\leq 3/4$ of undetected flaw in proof.

Figure 4.2. Summarizes cryptographic algorithms discussed in §4 and their costs (except for multiparty protocols – see table 4.3). All numbers and messages are assumed to be N bits long except that the key-lengths in secret key cryptography are K bits. "Modexp" means a modular exponentation (computing $a^b \bmod c$, or if we are doing elliptic curve cryptography, then this means an exponentiation inside an elliptic curve group); "AES" means an AES-like secret key encryption or decryption (of a variable-length message §4.5); "mul" means a modular multiplication $ab \bmod c$ (or group operation $a \otimes b$ inside an elliptic curve group); "ZKP" means "zero knowledge proof"; and $\lg x = \log_2 x$. ▲

Deal out shared secret: $O(T^2 + TQ)$ arithmetic operations mod P.
Same but verifiable: same but also compute S hash functions twice.
Same but instant-verification: $4S + 2T + 2$ modexps.
Reconstruct secret: $O(T^2)$ arithmetic operations mod P.
Add two shared secrets (in verifiable shared-secret form without reconstructing): $O(S)$ arithmetic ops mod P.
Multiply two shared secrets (ditto): poly(T, S) modexps.
ZK verifiable multiparty computation of logical circuit: $O(GS)$ modexps (or spot dishonest party).

Figure 4.3. Summary of multiparty "shared secret" cryptographic algorithms discussed in §4 and their costs. (We do not count prover-verifier interactions in ZKPs as "multiparty.") Same conventions as table 4.2 and also: there are S sharers, among whom $> T$ must collude in order to get enough information to know any secret. The "logical circuit" being simulated has G logic gates (each 2-input AND or 1-input NOT) and W wires (including inputs and outputs). ▲

5 Voting – realization of possibility

With the zero-knowledge and secure multiparty computation technology of §4, wondrous feats become, in principle at least, possible. In particular, we can now see almost instantly that election schemes exist satisfying (almost) all our desiderata.

We'll now make that existence argument, without trying (for now) to produce practical schemes. We are going to show how to run an election as follows:

1. Before the election starts, there is a pre-posted publically readable list of all legitimately-eligible voters.
2. During the election, voters provide their votes to the election authority EA. We assume these voting-and-communication processes are private, i.e. not seen, heard, nor recorded by anybody else.[55]

[55] A voter who videotaped himself during the act of simultaneously voting and exhibiting a copy of a message he had one minute ago received from a vote-buyer might immediately afterwards be able to sell his vote to that buyer. A vote-coercer could similarly have planted a hidden camera inside the voting booth, or could simply be physically present in the booth at the time... We shall regard all these possibilities as *collusions* between

3. Afterwards, the EA combines the votes (it does not matter which vote-combining method we use, so long as it is a publically known polynomial-time algorithm) to produce the election results, and announces them.

in such a way that unless there is a 3-way *collusion* between voters, the EA, and vote-buyers/coercers, then

1. Anybody can verify that only legitimate voters voted, and each voted at most once, and that no votes were faked or altered, and no votes were destroyed.
2. Anybody can verify that the correct election results were announced.
3. The whole election and verification procedure involves only a polynomially large total amount of computational work, i.e. both are "feasible."
4. No vote-buyer can know (with any amount of confidence beyond their unsupported assertion) what any voter's vote was, since no voter can prove how he voted unless an exponentially large amount of computing is done ("infeasible") or unless such widely-regarded-as-hard problems as large discrete logarithm problems, are somehow actually soluble in subexponential time. (I.e.: both vote-buying and vote-coercion are "impossible.")

This is accomplishable almost entirely with zero-knowledge proof technology. If, further, heavy use is made of secure multiparty computation, even stronger security guarantees become possible.

How to do it: To vote: Each voter encrypts his vote using a public key cryptosystem, so that the resulting vote is decryptable only by the election authority EA. Then he transmits the encrypted vote M to the EA, which encrypts it *again* and sends it back to the voter. Both use randomized Elgamal encryption and the method of §4.16 to allow the voter to be confident the EA really has re-encyrpted M, but without the EA revealing its encryption or decryption key. Finally the voter dates and signs this vote, proving that only he could have created it (remember, all can verify any signature using that voter's public verification key available on the pre-posted legitimate-voter list), and transmits it back to the EA, which then immediately signs it itself and posts it on the bulletin board – next to that voter's name on the pre-posted list. Both the EA and the voter must agree on the date and the time of day, which is incorporated into the doubly-signed vote (if not, one or the other will refuse to sign).

The EA's voting machine can then print out a receipt (containing the same information it is posting to the bulletin board, printed as "bar code") which the voter can scan to confirm its validity and then take with him as he leaves. The voter can confirm his vote has been received by inspecting the bulletin board immediately afterwards; if the vote was not posted, the voter can come forward with his receipt (which, since the EA signed it, must represent a legitimate vote) and complain, demanding that his vote be posted (which could then be done by scanning that receipt). These paper receipts would allow a later "recount" even if the entire electronic part of the voting system were destroyed (after rebuilding it).

The task of decrypting those votes, verifying that each voter signed them, verifying that no voter double-voted (or anyhow, discarding all but the last-dated one among multiple votes having the same signature), and that only legitimate voters voted (i.e. only those whose signatures are on a prepublished list of "valid" voters), then using the decrypted votes to compute the election result obviously is[56] in the class NP, indeed as far as the EA is concerned in the subclass P, since the EA knows all the required decryption keys. Therefore, by theorem 4.13, it is possible for some Prover (namely EA) to do them, thus computing the election result, while at the same time producing a zero knowledge proof of validity. Then anybody[57] can run the verification protocol to verify that the Prover really *did* compute that result correctly – although without thereby gaining any knowledge of what those votes were (aside from the knowledge inherent in everyone's knowledge of the announced election result)!

Why it works: Because everything is a zero knowledge proof protocol, everyone is confident that the EA has correctly transformed the input (the publically posted encrypted votes) to the output (election results).[58] Because the input is encrypted and only the EA has the decryption (and one of the encryption) keys, nobody but the EA knows what that input is. A voter wishing to prove to a vote-buyer or vote-coercer that he voted in some way, is unable to demonstrate regeneration of his publically posted vote because he has only some of the info (pad bits and encryption key #1) required to do it, not the rest (encryption key #2). (Also, no two posted votes will be identical – even if the voters voted identically and agreed to use the same encryption procedure and same random bits – unless the EA's re-encryption employed the same random bits for both voters, which would not happen except with a 3-way EA-involving collusion.) So unless the voter and vote-buyer can get key #2 from the EA (3-way collusion), the vote cannot be revealed. (Actually the EA could simply reveal the decryption of the vote to the vote-buyer/coercer alone – 2-way collusion – but since any buying or coercing also requires the participation of the seller/victim, this could be regarded as a 3-way collusion.)

Even in the event of such 3-way collusive vote-selling/coercing, those votes, whether bought, coerced, or honest, *still* will verifiably be combined correctly, and it will still be impossible for anybody to vote more than once, nor for any vote to be produced by a illegitimate "voter" not in possession of a valid signing-key. It is for this reason we refer to the EA as "*semi*-trusted"; even if it is completely corrupt it still is not capable of faking the election, although it is capable of allowing (or participating in) vote-buying, vote-

the EA and either the voter or buyer.

[56] With all commonly used voting systems, anyhow.

[57] In practice, just one, or a few, agencies should do this, but do it fully publically and do it with input about their random choices got from many members of the public and/or via a clearly-random source such as those machines which produce winning lottery numbers.

[58] Or confident the EA lied. If the EA is going to lie despite knowing that lie will be detected, they equally well could simply refuse to perform the election *at all*, or refuse to use our protocol, and/or just kill everybody they dislike and declare the "election" over. Or the EA could refuse to pre-register voters they dislike, or refuse to let voters they dislike vote at all (and instead jail them), or preregister a large number of artificial "voters," or jail all verifiers. So a certain amount of trust in the EA is unavoidable.

coercing, and all sorts of general mayhem, and is capable of refusing to hold the election at all.[59]

Verdict: We shall not bother to describe how to make this scheme efficient enough to make it practical (although we could). That is because semi-trusted election authorities are just not good enough. An EA with the capability of knowing everybody's vote is just too powerful.[60] The only reason we've described this scheme is to make the reader appreciate the power of zero knowledge techniques and their usefulness for election schemes. So we now instead shall devote ourselves to considering how to *prevent* the EA from knowing anybody's vote, while *still* forcing it to produce the correct election results – with proof it has done so. (Or, the EA can refuse, in which case everyone will know it.) We shall call such elections "fully secure."

In particular, §7.3 shall describe a version of the present scheme but now upgraded to full security, and how to make it efficient.

5.1 Election hashes

In the scheme above, and also in those of §7.1-7.3, where all votes are posted on a public bulletin board, and also where a list of all eligible voters (with their public signature keys) is pre-posted, there is a simple way to prevent alteration of these lists. It is also useful for allowing those downloading copies of the list to be sure they have done so without error.

The hash. Sort the list of votes (or voters) into lexicographic order and then run it through a secure hash function. The result is a fairly short bit string (32 bytes) that serves as a fingerprint for the entire list.

It is effectively impossible to alter the list by even one bit without destroying the validity of this fingerprint. (By computing many fingerprints, one after each time new information is added to the list, it becomes possible for somebody who keeps receiving those fingerprints to become confident that the list never shrunk.)

6 Where are we now?

We have made it clear that secure voting is possible in principle, and in a purely theoretical sense it solves the problem. (At least if a semitrusted election authority were considered good enough.) However, in a practical sense, merely the fact that a polynomial-time algorithm for something can be written down, is not good enough. We *apparently* would also need:

1. The verification protocols must be *efficient in practice*,
2. We need all computer code to be *clearly bug-free*,
3. We need all computer *hardware* these codes run on, also to be clearly bug-free.

Those standards are very difficult to achieve, especially simultaneously. In fact, considering the need to certify the hardware, and to certify the certifiers, and the compiler used to help create the software, and the operating system, et cetera, the latter two tasks seem essentially *unachievable*! Furthermore, even if they were achieveable, the common man in the street would be unwilling or unable to go through these certifications himself, in which case their effect would largely be moot.

But actually, we do not need those 3 things. Before giving up in disgust, consider this: really, it is *not* necessary to be sure the prover's software and hardware work – because the verifiers will verify that it produced the right result! And it is *not* really necessary to be sure the verifier's software and hardware work, because many different watchdog groups, all with independently created verifier hardware and software, can choose to be verifiers. As long as enough of them can be got to work, we are happy. (Further, it is easy to create artificial right and wrong "provers" to use to test the verifiers during their development, allowing a high degree of practical confidence to be attained. Again, if enough independent groups succeed in creating such test-provers, we are happy.)

So the only requirement we *really* need is efficiency in practice, and *that* is an achieveable goal.

At that point we would have a truly remarkable and unprecedented level of voting system security. In the entire history of humanity, nobody has ever been able to solve any integer factoring problem for a product of two randomly chosen 300-digit primes, or to solve any random discrete logarithm problem based on a 300-digit prime. In contrast, all present-day attempts intended to increase voting system security, such as bipartisan election observers, locks on cabinets containing votes, hiring trustworthy people to run the elections, and so forth, have many times been subverted, undoubtably often without detection.

7 The four main approaches to efficient and fully-secure elections

The four known approaches are

1. Schemes based on homomorphic encyrption (§7.2).
2. Schemes based on mixnets (§7.1).
3. Heterodox schemes (§7.4).

[59] One interesting means of doing so would be for somebody at the EA, and a group of anarchist voters, working in collusion, to produce many legitimately signed voting receipts corresponding to unposted votes. The anarchists could then use these receipts to "prove" the EA corrupt and thus invalidate the election.

[60] David Chaum's election scheme [35] is an example of a method involving a merely semi-trusted EA (although Chaum seemed to believe otherwise); hence it in our view is unacceptable. That is because Chaum's method involves a voter, as he votes, getting two receipts from a printer attached to the voting machine, one of which then is destroyed in a publically visible shredding machine. Unfortunately, there is nothing stopping the EA's voting machine from remembering Chaum's "destroyed" information, in which case the EA would know everybody's vote. Chaum's scheme is also vulnerable to voters who refuse to destroy one of their receipts (if they keep both, they can use them to sell their vote); although Chaum suggests a way on his pp.39-40 for the EA to discourage that by causing those receipts to be invalid for the purpose of later use in proving their vote, we respond that (1) Chaum's discouragement-method would not affect voters uninterested in confirming their vote was truly accepted/posted by the EA, but instead only interested in selling their votes, and (2) his discouragement method anyway could be defeated by a voter armed with either a scissor or a camera, (3) or by any voter colluding with the EA. Chaum's scheme also is based on a mixnet (for criticism of mixnet schemes see §7.1) and is vulnerable to the same "anarchists" as in footnote 59.

4. Schemes based on secret sharing among several mutually distrustful election authorities (§7.5).

We'll discuss exactly what these ideas are, later. For now, let us briefly point out serious deficiencies in all of these ideas – not all of which had been pointed out by the cryptographers who invented and developed them!

1. While homomorphic encyrption schemes are excellent if votes combine additively, they simply cannot be used if the election results arise from a *non-additive* method of combining the votes, such as Hare/Droop STV (single transferable vote) schemes [110][161] or my own "reweighted range voting" [150].[61]

2. Many mixnet schemes depend on the idea that if all the votes are somehow posted in *random* order, with that order being unknown to anybody, then that means that nobody can deduce from that posting which voters produced which votes. While that is a pretty good assumption if the only possible votes are "yes" and "no," it unfortunately is flat wrong if we are using an election method such as *range voting* [149] or *asset voting* [151] in which each vote involves many real numbers. By altering low-significance bits in those real numbers a voter may effectively uniquify his vote, enabling him (after the posting of the scrambled vote list) to sell his vote to a buyer who had pre-chosen it.[62] Even in all-integer election schemes such as STV, in which each vote is a preference ordering (permutation) of the n candidates, and these permutations are combined in a nonadditive way to determine election winners, there can be a considerable amount of freedom in each vote, since there are $n!$ possible permutations. A voter could permute the hopeless candidates who are believed to have no chance of winning, in an attempt to uniquify his vote.[63] This rules out mixnet-based schemes for use with *all* of (what I consider to be) the best available multiwinner election methods!

Another problem with mixnets is that, as they operate, they necessarily perform an enormous amount of communication (1000s of bits per voter) between the different parties running the mixnet. That could make them undesirably expensive, unreliable, vulnerable to "denial of service" attacks, or simply infeasible.[64]

3. On the other hand, a possible *advantage* of mixnet schemes it that they can prevent us from knowing *whether* somebody voted (see 2c in §2) thus preventing the problem of "coerced abstentions." (Actually the usual mixnet schemes do not hide that information, but the possibility at least is there.) Meanwhile the usual homomorphism-based schemes make it obvious who voted and who didn't.

4. Many schemes based on secret sharing among several mutually distrustful election authorities may not be politically realistic. Is it really realistic to expect there to be several, *independent*, mutually mistrusting noncolluding election authorities when we consider the fact that in practice, all will involve many bribeable employees – and that there is only one government? Perhaps political parties could do the job – but would that not create a severe disadvantage for unaffiliated candidates? And is it really realistic to expect each voter to communicate his vote to *all* those authorities?

If the "authorities" were the candidates themselves (on the view that *they* would be mutually mistrustful): then (a) it seems unreasonable to suppose that each candidate (especially ones not affiliated with a political party) would have enough computational and comunications resources to handle the job, and (b) suppose one political party decided to sponsor a large number of "competing" candidates to run – who would then all collude during the vote-counting process?

The idea of secret sharing is a good one, but in practice, to be maximally acceptable and realistic, such a scheme should involve a highly asymmetric role being played by the sharers. One "big" sharer ("the" election authority) should do almost all the work and all the communication with voters. The others should do almost nothing and store almost no information and consequently should require very few employees and hence be both difficult to corrupt and truly likely to be mutually independent and mistrustful.

What we shall do. First, we explain and disparage mixnet schemes in §7.1.

Second, in the event that an additive vote-combination method is being used[65] we propose in §7.3 an excellent scheme based on a combination of homomorphic encryption and secret sharing. It may be regarded as a fully-secure version of the semi-trust scheme of §5. It allows recounts based on paper ballots if the internet should fail and the algorithms are both efficient and highly (and trivially) parallelizable with tiny communication needs. The secret sharing is highly asymmetric in precisely the fashion we just said was desirable.

Third, suppose a *non*additive vote-combining method is to be used, such as Hare/Droop STV [110][161], reweighted range

[61] Many election schemes which initially may seem non-additive, such as Condorcet and Dodgson, may be reinterpreted as additive. However, I see no way to do that for RRV except by working in a possibly ultra-high dimensional space in which the voter, essentially, provides weighted copies of his vote for all of an enormous set of possible weight factors. In an election with 135 candidates (such as the 2002 California gubernatorial election) and 6 winners (many 6-winner districts occur in contemporary democracies) this would mean providing 360,343,288 copies, while if there were 60 winners then about 10^{39} copies would be needed. In STV this situation would be even worse because the weight factors are not predictable in advance in the sense that, if the number of voters were not known in advance, then there would be an *infinite* set of possible future weights for any given vote.

[62] In plain 1-winner range voting, the kth coordinates of each vote-vector could be permuted via a different permutation from those used for the other coordinates, which reduces the magnitude of this problem but certainly does not eliminate it because a voter could still point out to a buyer the amazing coincidence that each of his preselected reals appeared *somewhere* in each list. Because 1-winner range voting is an *additive* vote-combining method, bit-splitting schemes (see §7.1) and perhaps some kind of "glorified" mixnets which also transfer some random fractions of each real randomly to other reals (i.e., not merely performing a permutation) might be useable to defeat this objection, but those ripostes would not be available if a nonadditive vote-combining method such as RRV were used, because then a true permutation would be required and all real coordinates of the vote-vector would have to be permuted by the *same* permutation.

[63] Since $13! \approx 62 \times 10^8$, almost all most voters could expect to succeed at this attempt in a 16-candidate election with 10^8 voters where only 3 candidates were regarded as having good chances to win.

[64] Also, if a *truly* random permutation is desired, then *all* the votes must be available before starting the mixnet, which could lead either to serious delays or to the possibility of "late" voters of various kinds selling their votes.

[65] These include plurality, Borda, Dabagh, Condorcet, approval, range, asset, Dodgson, and eigenvector voting [107][149].

voting [150], or sar-range [153]. Then we shall demonstrate how, in principle, these too may be made fully secure with the aid of secret-sharing and secure general multiparty computation. Unfortunately this scheme involves a fairly symmetric role for the sharers and requires an enormous amount of communication both between them and the verifiers; each voter must communicate with each sharer; and there is an enormous amount of total computation. These problems are severe enough to render this method infeasible with present technology.

7.1 Mixnets

A *mixnet* is a multiparty computation-and-communication protocol that causes a large number of input messages to get shuffled into a random order in such a way that every party (as well as external verifiers) becomes confident that a shuffling was performed, but no party (nor even any t-element subset of corrupt colluding parties, provided that t is not too large) has any idea *what* the shuffle-permutation was.

In the original proposal by Chaum [34], the mixnet was the net effect of a sequences of individual permutations, or *mixes*. The individual messages remain in various encrypted changing forms throughout mixnet operation, only becoming decrypted at the final stage, and hence there is in general no way for anybody to know which output message arose from which input message. Chaum achieved that by initially multiply encrypting each message with the public keys of all the mixers it later will traverse (in reverse order); each mix then decrypts the messages it permutes, with the final mix restoring the original cleartexts. By making each mixer perform a verifiable shuffle (§4.12) observers can become confident that each mix really is working as advertised.

If the mixers are operated by mutually distrustful and hence noncolluding parties, then although some parties may know some of the component permutations, nobody can know the final product permutation. Several proposals have been advanced to allow this to be reasonably efficient [1][2][67][78][92]. [66]

We now give our own mixnet proposal, which is somewhat similar to a proposal of Abe [2] and which sacrifices perfectly-uniform provable randomness of the permutation to achieve high speed, simplicity, and to allow it to provide non-interactive validity proofs (to reduce communication requirements). This sacrifice seems not to matter for the purposes of voting schemes in large elections.[67]

New, maximally-practical, mixnet scheme: Suppose there are N items to permute. Each individual mixer will send (after re-encrypting it) input-item i to one of three output positions according to some fixed, public, $N + N$-vertex 3-valent bipartite expanding graph. Note: this causes at most 3^N permutations to be possible per stage, which is considerably less than the full $N!$. But it nevertheless seems entirely adequate.[68] Specifically, let us say that input i is routed to either output A_i, B_i, or C_i for some public known disjoint N-permutations A, B, C.

Each mixer's job is to provide a simple non-interactive zero knowledge proof that it did that. For that, it must, for each $i = 1, 2, \ldots, N$, ZK-prove that output A_i is a re-encryption of input i, *or* B_i is, *or* C_i is. This may be done by *or*ing together (using the *or*ing technique of §4.17) three re-encryption ZK-proofs of the sort in §4.16. (These proofs are all non-interactive.)

The net effect of many consecutive such mixers, each choosing a random accessible permutation, is that each input is routed to a random output. (Note: because of eigenvalue bounds, a random walk on an expander graph is known to "rapidly mix" toward the uniform distribution.) Since correlations among the random walkers plainly seem small if the graph is expanding, the conjectural net effect should be a random permutation, with quite close approach to the uniform distribution after $O(\log N)$ random mix stages. Assuming we were planning on using that many mix stages anyhow, this amount of computation is optimal. ▲

Incidentally, in the above scheme the graph need not be the same each mix-round. Instead, each mixer could choose (and announce) its own graph randomly each time (3 random permutations are disjoint – and expand – with probability bounded above a constant, so simple trial suffices as a graph-construction method) which would have the advantage that each mixer could produce an *exactly* uniformly random shuffle-permutation, albeit with somewhat greater proof-length.

A typical voting scheme based on blind signatures and mix nets [94] proceeds as follows.

1. Voter gets ballot from Manager.
2. Voter fills in the ballot with his vote in a legitimate way.
3. Voter gets it blind-signed by $> T$ out of S total Administrators. In order to convince them to provide their signature, the voter will have to zero knowledge prove that his ballot is legitimately filled in, and will have to provide proof of his identity (e.g. by proving knowledge of a discrete log posted under his name on the pre-published list of eligible voters). Each administrator, whenever she signs a ballot, will also post, under that voter's name on the publically-viewable eligible voter list, a note, signed by both the administrator and voter, saying "I blind-signed this voter's ballot." (This prevents the voter from voting twice.)

[66] It might conceivably also be possible to shuffle the votes using some trusted physical mechanism instead of a multiparty proof protocol.

[67] Abe's schemes as corrected in his second publication [2] *can* be used to generate an exactly-uniformly random distrituion of shuffle permutations. Abe employs a Benes "telephone network" to shuffle N items via $2\log_2 N$ stages, each stage consisting of $N/2$ "crossover boxes" which can either swap its two data $A \leftrightarrow B$ or not (while re-encrypting all the data in both cases). His shuffler chooses the permutation at random, determines (using a standard routing algorithm) how to achieve it using the Benes network, performs it using the Benes network, and finally provides ZK proofs that each Benes crossover box did perform as advertised (but without revealing which choice was made by each box). These Benes-box ZK proofs can be made from an OR of two ANDs each of two re-encryption ZK proofs. This all is somewhat more complicated than our scheme since Benes networks and routing algorithms are involved and there are further adjustments if N is not an exact power of 2.

[68] The exact count of accessible permutations is the permanent of the $N \times N$ adjacency matrix of the graph. By choosing a graph such that this matrix's *determinant* is exponentially large, we get an exponentially large lower bound on this count. Random 3-valent graphs will usually both be expanding, as may be confirmed by an eigenvalue test, and have determinant of order $3^{N/2}$; and explicit graph constructions should be creatable to do these jobs, avoiding any explicit necessity for eigenvalue or determinant checks.

4. Both the mixnet and the voter post a co-signed note to the bulletin board saying "I have submitted my blind-signed vote to the mixnet," and simultaneously voter submits that multiply-encrypted[69] ballot to an input of the Mixnet.
5. After all votes have been input and the polls close, the mixnet operates to permute the encrypted votes, thus anonymizing them.
6. If the number of mixnet outputs differs from its number of inputs, of if the mixnet produced a bogus validity proof for its operation, everybody knows it cheated.
7. Counter decrypts the outputs of the mixnet and posts the shuffled and blind-signed cleartext votes on a bulletin board for public view.
8. Now Counter (or anybody else) may combine them (discarding the ones with invalid blind signatures) to determine the election result, which therefore is seen by all to be correct.

The point of requiring $> T$ out of S administrators to sign each ballot, is that hopefully, although some administrators may be corrupt, at most T of them are. There are two ways administrators could be corrupt: they could refuse to sign a legitimate ballot of a voter they dislike, or agree to sign an illegitimate ballot. That implies that the largest we can make T is to have $S = 2T + 1$. Administrators could also try to create fake ballots, and, acting in collusion with an input-accepting part of the mixnet, substitute them for legitimate votes – but this would be impossible if $\leq T$ of them colluded.

We disparage mix nets, for several reasons. They cannot be used with any of the best multiwinner voting systems, because those systems permit uniquifying votes, in which case mixing cannot satisfactorily anonymize them.

It actually *is* possible for mixnets to handle range voting – despite the ability of voters to uniquify their range votes by encoding information in low-significance bits – by having the range-voter actually produce *many* 1-bit subvotes, one for each bit of the complete vote (and labeled with a description of which bit-position it is). These subvotes *are* highly likely not to be unique in any large election, which effectively would prevent vote buying, albeit at the cost of requiring a large increase in the amount of mixnetting and blind signing work. However, in nonadditive schemes such as reweighted range voting, votes have to be delivered at the mixnet output *as a complete package* and not as separated and unassociated bits. For this reason mixnets cannot handle either reweighted range voting or STV voting.

Another annoyance is that (at least if we are not willing to sacrifice perfect anonymity) mixnets require *all* votes to be available before they *start*. That could cause large delays.

In fact, mixnets seem to handle a no-larger set of allowed voting systems (at least among the commonly known ones [107][149][151][150]) than the homomorphic schemes of §7.3, and pay a very large performance penalty in cases where bit-splitting is required. These cases include range voting [149], the best 1-winner election system!

If bit splitting is not required, then mixnet schemes are perhaps 5 times worse from an efficiency standpoint than homomorphic cryptography based schemes (§7.3). Indeed, it is conceivable that they might become almost competitive in efficiency if certain compromises are made (such as only requiring shufflers to prove that they have shuffled *all but a few* of the votes, see [92] or §4.12). The devastating problem with mixnets is not their overall amount of *work*, but rather their immense interprocessor *communication needs*.

Short diatribe about parallel computer programs. To explain that: Parallel computer programs generally are a bad thing. They tend to be nonportable, and are usually hard to debug and unreliable because they can exhibit irreproducible behavior. They are vulnerable to attacks on the interprocessor communication system. The only *good* kind of parallel program is a sequential (i.e. *non*parallel) program that is simply run many times! These are easy to debug, run robustly, and their interprocessor comminucation needs are almost zero.

Three examples:

1. Consider adding up 10^{12} numbers. This is trivial to parallelize: 10^3 processors could each add up 10^9 of the numbers, and then report their subtotals to a final processor for final tallying.

2. Another example is a Monte-Carlo randomized search for snarks.[70] Many processors each search using same program but with different random seeds; whenever a processor finds a new better snark improving on the current snark-quality record, it reports it to a central record-keeper.

These were examples of the good kind of parallelism. They have simple programs, tiny communications needs, and the precise timing and sequencing of each communication is almost irrelevant.

3. A perfect example of the worst kind of parallelism is a mixnet – in which there is a vast amount of interprocessor (and processor↔external-verifier) communication, all of which must be precisely sequenced and all of which is centrally important to obtaining the final result.

Mixnets would unfortunately have to be heavily parallelized in large elections, both because of current technological limitations on computer speeds (cf. §3), and also because of the fact that they need to be run by several mutually distrustful *physically isolated* parties (if all parties colluded, then they could perform a known permutation and all authors of all votes would be known to them).

In an national election, i.e. an adversarial environment possibly featuring well-equipped and heavily funded attackers, this amount of parallelism combined with the need for a huge amount of long-distance communications, becomes a complete nightmare.

The requirement in this system for the voter to contact many different independent Administrators, each of whom must perform a large amount of communication and computation, seems unrealistic, impractical, or undesirable. The requirement for verifiers to conduct a many rounds of heavy interaction with the (heavily parallelized) mixnets also seems unrealistic, impractical, too-easily attackable, or undesirable.

Finally, the mixnet scheme we have just outlined is subject to the criticism that the precise subset of administrators that

[69] The encryption is undoable by the net effect of each mixer's individual decryption followed finally by the Counter's decryption

[70] Rare, high-quality objects of some sort.

sign your vote, can go a long way toward identifying the voter. If there are 31 Administrators and each voter must have 16 sign his ballot, then since $\binom{31}{16} = 300540195$ that would be more than enough to uniquify ballots in a 10^8-voter election. A cabal of colluding adminstrators could thus go a long way toward enabling vote-selling and/or coercion; and even without precisely identifying voters, they could still identify certain statistical trends they could use later to punish entire voter groups.

7.2 Homomorphic cryptography and its uses

An encryption function $F(M)$ is said to be *homomorphic* if $F(M_1)F(M_2) = F(M_1 + M_2)$, that is, if multiplying two encrypted messages is the same as encrypting the sum of the messages, for some appropriate notions of "sum" and "product."

The "one way communication" version of Elgamal public key cryptosystem from §4.8 can easily be made homomorphic as follows.

To encrypt a small nonnegative integer v, let $M = g^v \bmod P$. Then its encryption is $(k, c) = (g^r, g^v y^r) \bmod P$. The elementwise product (mod P) of two such encryptions is $(k_1k_2, c_1c_2) = (g^{r_1+r_2}, g^{v_1+v_2}y^{r_1+r_2}) \bmod P$ which is an Elgamal encryption of $v_1 + v_2$.

All this also may be readily transplanted into the elliptic curve group framework instead – and please do, since ECC systems provide superior security (§3.2).[71]

It is also possible to devise homomorphic public key systems based on RSA or generalizations thereof [47][4],[72] but the security of those ideas are worse since they depend on the hardness of factoring rather than the hardness of the discrete log problem in elliptic curve groups. Therefore we shall not discuss them.

Since in many voting schemes (see footnote 65) the method of combining all votes can be regarded (essentially) simply as addition, homomorphic encryption permits us to add up all the votes *without ever decrypting them* (indeed, this adding can be done by somebody who does not even *know* the decryption key) until at the end we decrypt the vote *total* [41][42]. Unfortunately, anybody with the power to decrypt the vote totals would (at least in this Elgamal scheme) also have the power to decrypt the individual votes. Furthermore, when the vote total T is "decrypted," we actually get $g^T \bmod P$, which is not quite the same thing as T itself. The only way to get T itself is then to solve a discrete logarithm problem, which fortunately is not difficult if we know that T is small (e.g. since all votes are small integers and there are a small number of voters) since we may try all possible T exhaustively.

Because range [149] and asset [151] voting involve the addition of *real* numbers rather than small integers, they might be thought incapable of being handled by homomorphic encryption. However, they actually can be handled if we employ fixed-point reals (i.e. integers), provided P is much greater than the number of voters times the maximum integer we would permit in any vote – a requirement certainly satisfied by a 300-digit-long prime P. The only problem is that with fine enough "grain sizes" for our discretized reals, the search for the solution of the discrete log problem might be a much larger search! Fortunately that difficulty seems soluble in practice using the "baby step giant step" or Pollard-lambda methods (§4.3) of solving discrete-log problems in sublinear time. Specifically, if there were 10^{10} voters (greater than present world population!) each using 5-digit reals, then we would need to find a discrete logarithm between 0 and 10^{15}. Exhaustive search of 10^{15} possibilities is not feasible, but either the baby/giant or the Pollard method would solve the problem in $\approx 10^8$ steps, each "step" involving a modular multiplication or inversion, and perhaps a table lookup. That would be entirely feasible in less than a day on one standard personal computer.

A different approach (which also would work) would be to add up the *single bits* in each range vote real number in separate "elections," i.e. one for the most significant bit, one for the next most significant, and so on. The bit-splitting method would usually be less efficient because it requires a large number of modular exponentiations *per vote* to encrypt all the bits in that vote separately, whereas the work in the Pollard/baby/giant approach is *sub*linear in the number of votes (and also enjoys a smaller constant factor since the operations in its inner loop are modular *multipications* rather than *exponentiations*), and also if any zero knowledge proofs of interval-membership were devised that were sublinear in the number of bits (see §4.22).

Unfortunately, homomorphic encryption methods seem unsuitable for running elections in which the vote combination method is *not*, at its core, merely addition. The following seem (if there are achievably large numbers of candidates and/or winners) to be unhandleable: Hare/Droop STV [110][161], reweighted range voting [150], and sar-range [153].

7.3 A practical secure election: Homomorphic repair to §5's semi-trusted scheme

We now can make a few alterations to the semi-trusted election scheme of §5 to make it fully secure. As before, paper vote-receipts allow recounts if the internet fails. First, let us make all encryptions be homomorphic Elgamal and have the election authority EA *not know* its own decryption keys.

Thanks to homomorphism, this in no way prevents EA from adding up the votes, or proving to each voter it has correctly re-encrypted her vote. (Furthermore, anybody else is free to carry out exactly the same computation as the EA, hopefully reproducing exactly the same result, and by having many different processors, each computing a subtotal of the total sum, we *trivially* can parallelize this computation as much as we want – although that would not even be necessary since no modular exponentiations – only plain group operations – are required to do a homomorphic addition causing the entire summation to be rapid[73].) But it does prevent EA from

[71] Notice also that taking the pth power of an Elgamal homomorphic encryption is the same as encrypting p times the original number.

[72] Acquisti's scheme [4] employs *both* the Paillier public key encryption, which depends on the hardness of integer factoring *and* homomorphism *and* mixnets. We would therefore not recommend it on efficiency grounds.

[73] "Adding" up 10^{10} votes (more than present world population) where each addition is really a modular multiplication with a 512-bit modulus, would require 25 hours on a *single* Pentium-II/200MHz processor.

revealing any of the votes, causing the election to be fully secure (since we already saw that the EA's collusion in revealing votes or decryption keys was required to compromise the system – this collusion is now impossible).

In the original scheme, since EA could decrypt all the votes, EA had no difficulty in assuring itself that every submitted vote was correctly and legitimately formatted. It can no longer do that. Therefore, it is now necessary for each voter to provide, in addition to merely her vote, also a zero knowledge proof that the vote was correctly and legitimately formatted. Such proofs can be based on the methods of §4.17 and §4.22.

With only these minor alterations, we now have a fully-secure verifiable election scheme. There is only one problem: the sum of the votes (ready for use to produce the election result) is only output by EA in *encrypted* form – and while everybody can convince themselves that is the encryption of the correct sum, neither EA (nor anybody else) knows the decryption key!

The solution to that is to have the decryption key K be known, *not* by any one entity, but in fact partially by several mutually mistrusting entities: Each entity i knows K_i where $\prod_i K_i = K \bmod G$ (where G is the publically known order of the group g and h are from). The entities can work together to decrypt an Elgamal 2-tuple $(g^r, h^r M)$ to find M from the publically known g and $h = g^K$. (Here r is random and not known to anybody.) The objective is to compute h^r from g^r by raising it to the power K. That may be accomplished by having each entity i raise it to the power K_i, successively, after the election is over.

The initial generation (before the election starts) of $h = g^K$ is similarly accomplished by the entities successively raising g to powers K_i that they choose randomly at that time. During the second go, the entities must provide zero knowledge proofs (via the method of §4.16) that they are using the same power K_i that they used on the first go!

In this way, *nobody* ever knows the Elgamal decryption key K – but we may still decrypt the election result!

Unless all the keyholding entities (and EA) collude, all votes will remain forever private. Since these entities are assumed to be mutually mistrustful (e.g. they might be: the major parties and citizen's "watchdog" groups), at least some of them will not collude. Note that the vast bulk of the computational and communication work is done by EA in this scheme. The keyholding entities only act once at the very beginning, and once at the very end, to perform a few modular exponentiations each, i.e. a small amount of work independent of the number of voters.[74]

Because the EA does not know *de*cryption keys, only *en*cryption keys, it can distribute those encryption keys far and wide to its many voting machines without fear that any important secret will be compromised. In fact these keys can and should be *publicized*.

Because of this, all communication between the voter and the EA during voting can *really* be a communication between the voter and the EA-owned voting machine sitting next to him – there is no need for any voter to communicate with anyone over any long, tappable wire.

Summary of entire proposed election procedure.

1. The s keyholders randomly generate their secret partial-decryption keys K_1, K_2,..., K_s and use them to produce public encryption keys g and h^K where $k = \prod_{j=1}^{s} K_j$.
2. Voter generates his vote as guided by publically posted ballot. Vote consists of integers v.
3. Voter homomorphic-Elgamal-encrypts (§4.8) his v's with public key K.
4. Voter transmits encrypted vote M to EA.
5. EA re-homomorphic-Elgamal-encrypts vote using public key K_E. That is, if the original encrypted vote V was $M = (g^r, h^r i^V)$ with r randomly chosen by the voter and where i is a public constant fixed random group element, then the new one is $M = (g^s, h^s i^V)$ where s is random and is the sum of the voter's r and the EA's different r (call the latter q).
6. Voter and EA jointly produce zero-knowledge non-interactive proof of the validity of the re-encrypted vote (e.g. using methods of §4.23 in a joint version as in the end of §4.20). Then EA adjoins the date to get M', which is sent back to voter.
7. EA alerts voter if this ZK-proof was not valid, in which case refuses to accept vote.[75]
8. EA privately supplies voter with zero knowledge proof of discrete log equality as in §4.16 to show $q = q$, i.e. that it has produced a valid Elgamal re-encryption, and this proof should be a "designated-verifier" ZK-proof, as in §4.19 designed to convince that particular voter *only*.[76]

[74] It might be objected that our system would be vulnerable to attacks on the keyholders. Killing even a single keyholder – or anything that caused him to refuse to cooperate – would force the entire election to be called off. The keyholders are the one irreplaceable element in our scheme. We can respond to that objection. First of all, keyholders, since they have so few responsibilities beyond remembering their keys, can be well protected. Second, the ith "keyholder" could in reality be a *group* who collectively generate and share K_i as a shared secret as in §4.11. As long as the majority of this group survived in a noncorrupted form, the K_i would be regenerable, and no minority-subset of the group would be capable of revealing K_i. Note that the techniques of §4.11 permit the group to raise a number to the power K_i (actually, to a power that is a fixed public constant multiple of the secret, but this makes no difference; this is due to the distinction between the quantities we call I_k and L_k in §4.11) in spite of the fact that no individual member ever knows K_i; and this is the only purpose for which they employ K_i.

[75] EA could always refuse to accept that voter's vote (based on a general prejudice against him) regardless, there is nothing that can be done about that; but it is entirely possible to have several EAs, all simultaneously collecting votes, in which case hopefully one will not be prejudiced against that voter; and a stiffed voter would be able prove to all that his vote never appeared on the bulletin board, and then demand redress. It is probably best that different EAs use different signature keys so that a cheating EA can be detected; this results in a (small) reduction in voter privacy.

[76] This idea of using a designated-verifier ZK-proof of re-encryption in this way – so that the voter knows his unaltered vote was posted on the bulletin board, but is unable to convince any would-be vote-buyer of any information about the *contents* of that vote – may be new. The EA could refuse to provide this proof, in which case the voter then would refuse to sign the vote (and again since the EA could *always* refuse to accept that voter's vote, there is nothing that can be done about that standoff). Further, the EA could collaborate with *both* the voter and a vote buyer to allow that voter to prove the value of his vote to the buyer and hence sell his vote. However, that would be impossible if only two of these 3 parties collaborated. This flaw also seems inherently unavoidable since if the EA wanted to collaborate with a vote buyer it could, more simply, just allow that buyer to be (tele)present in the voting booth with the voter.

9. Voter confirms date and re-encryption validity, then signs M' and sends back to EA.
10. EA alerts voter if his signature was not valid.
11. EA signs it also.
12. EA posts resulting twice-signed, validity-self-proving, and dated Elgamal-doubly-encrypted vote to world-readable bulletin board next to that voter's name on pre-posted list of all eligible voters.[77] EA also prints two paper copies of the post (as "bar code") giving one to the voter and keeping one for backup records.
13. Voter may scan his paper receipt and check EA-signature to verify its validity.
14. (Voters can vote multiple times but only their last-dated posted vote will be used. Voter can examine the bulletin board to verify their vote was posted.)
15. Once all votes have been acquired and posted, EA (or any other external totaller) uses the most-recently dated posted versions of each voter's vote (among those with valid voter- and EA-signatures and valid dates) and homomorphically-adds them up (simply by elementwise multiplication of the Elgamal 2-tuples) to get Elgamal-encrypted version (with public key K) of election totals.
16. The s keyholders successively partially Elgamal-decrypt the election totals using keys K_1, K_2,..., K_s (zero-knowledge-proving as they do so, that they are using the same K_j's as exponents as they did at the beginning, see §4.16 for the proof technique), and broadcast the resulting exponentiated forms i^T of the election totals T.
17. Somebody uses Shanks baby/giant method or Pollard-lambda method (§4.3) to determine the plaintext election totals T and then broadcast *them.* (These final broadcasts may be immediately confirmed by anybody.)

This is a very elegant solution. The total amount of work involved by the voters and EA amounts to about $25 + 11\ell$ modular exponentiations per voter-supplied-number assuming the naive interval-membership ZK-proof of §4.22 (in the Elgamal version of §4.23) is used and that each number supplied by the voter in his vote is ℓ bits long. (Better interval-membership ZK proofs, if they became available, could speed things up by replacing the ℓ by some much smaller quantity for large ℓ, such as $O(1 + \log \ell)$. Note this reckoning has not counted work done by the external verifiers.) This should require less than 1 second of processing per voter-number, even on slow computers. Assuming each voter is entering his vote into a machine which itself is a (low speed) computer, the voter will be unable to enter his vote as quickly as it will be processed. Almost all the work is involved in generating, encrypting, signing, and zero-knowledge-proving the votes (and verifying those signatures and proofs); once this is done the task of actually homomorphically adding up the votes is comparatively trivial.

The total cost of enough computer power to perform 1 second worth of processing per voter (and get it all done in 1 day) is (assuming \$1000 per computer) about 1 penny per voter. So, obviously, this system is buildable now with currently available technology.

7.4 Heterodox voting schemes

In this section we'll discuss schemes which do not fit neatly into the "homomorphic" or "mix-net" camps, either because they use both ideas, or other ideas.

Kiayias & Yung [99] suggested a very interesting and elegant, but ultimately unacceptable, voting method achieving

Perfect ballot secrecy: A voter's vote can never be revealed against his will unless *all* the other voters conspire against him.

Dispute-freeness: It is trivially apparent to any casual third party that all voters and the EA followed the protocol (or: if they did not, then it is apparent who cheated).

Self-tallying: Determining the election results from the publically posted data is fairly easy and can be done by anybody.

However, the Kiayaisis-Yung scheme suffers from three major disadvantages:

Quadratic storage: The amount of data generated and then stored on a public bulletin board is $5N^2 + O(N)$ data items for N voters. Each voter posts $5N + O(1)$ encrypted items. Even more posts are required if some voters later drop out. With 10^8 voters each voter would have to post about 10^8 data items, with over 5×10^{16} data items posted in all. With current technology, this is infeasible, so that their scheme could only be used for small elections ($< 10^4$ voters).

Vote buying: The scheme permits voters to intentionally reveal, and prove, their vote, thus permitting "vote buying."

Intolerable delays...: Their scheme involves a "preparatory phase" followed by a "vote-casting phase," and they allow voters to drop out between the two phases. But unfortunately, they require explicit identification of *exactly* which voters plan to drop out, *before* voting commences, so that the remaining parties may make appopriate adjustments (fairly expensively) to the quantities that had been published during the preparatory phase. That is unrealistic in large elections (e.g. some voters might die during their trip to the polling-place). Further, if some voters then cast illegal ballots (i.e. with false ZK validity proofs) then they have effectively dropped out, which would require the entire election to be restarted, with re-adjustments made each time that happened. The resulting amount of labor and time would still be bounded, because each time this happens, we have one fewer voter. But instead of requiring $O(N)$ work and communication per voter (which is already outrageously large), $O(N^2)$ could be required (and in fact would be required, if a constant fraction of votes were illegal) during order N different voting-again phases. This seems entirely unacceptable in practice.

It is possible to reduce their quadratic storage and work to linear by having only a sparse subset of their s_{ij} values be

[77] Consequently is is publically known who voted, disappointing advocates of "invisible abstention" property 2c from §2. However, we reply that in many election-tallying systems, such as "range voting," it is possible for voters to cast "null votes," functionally equivalent to not voting.

nonzero. This would come at the partial sacrifice of "perfect ballot secrecy" since now a much smaller set of voters could conspire to reveal i's vote, namely the voters j such that $s_{ij} \neq 0$ and who did not drop out. Indeed if all these were to drop out, then no conspiracy whatever would be required – voter i's vote would simply be revealed. This perhaps is an acceptable risk if the sparsity pattern is well chosen. It might also be possible to prevent vote-buying by use of re-encryption and designated-verifier validity-proving techniques, but that would come at the cost of eliminating "dispute-freeness" and "ballot secrecy" since disputes could now occur between the voter and EA. (Also a 3-way collusion between the buyer, seller, and EA would still enable vote selling as usual.)

But, I see no way to solve the "intolerable" problem, at least without a great deal of additional property-sacrifice, and unless and until it can be solved, the Kiayias-Yung scheme must be dismissed from practical consideration.

They have some nice ideas, however, and hopefully somebody will find a way to resuscitate them for some other purpose.

"Coercion-resistant" scheme by Juels, Catalano, and Jakobsson [96] as improved by Smith [152]:

In this scheme the identity of the voter remains hidden during the voting process. Each ballot contains inside it a "concealed credential" in the form of an encryption of a secret value σ known only to that voter, and an encryption of the vote itself, and a ZK validity proof. The tallier discards all votes with bogus validity proofs. To ensure that ballots are cast by legitimate voters, the tallier performs a blind comparison between the hidden credentials on the (scrambled-order and re-encrypted) list of votes, and each member of a pre-scrambled and pre-encrypted list (both scramblings and re-encryptions are got by putting the lists through a mixnet) of genuine also-encrypted credentials generated from the pre-posted list of legitimate voters. This comparison may be done with the aid of a "plaintext equality test" (§4.16). Of course validity proofs of all the mixing and plaintext-equality-testing are broadcast to all verifiers.

This allows the tallier to determine each vote's legality, and to prevent double voting, but without knowing any vote's author and indeed without it ever being revealed *who* voted. If several votes by the same voter are detected during tallying, then all but one of them is arbitrarily discarded (perhaps according to some predetermined policy such as keeping the chronologically last vote cast).

Finally, the votes in the final weeded list are decrypted and totalled.

JCJ employ several mutually distrustful talliers who cooperatively decrypt the final weeded votes (no one can do this decryption individually) and who cooperatively perform plaintext equality tests.

Vote-coercion attempts will not work because the coerced-voter could simply provide a vote with an invalid credential. Because plaintext equality tests are not performable by individuals but only by a threshold set of vote tallying authorities working in cooperation, the coercer cannot check the credential's validity. The official validity checks are only performed after mixing, so the coercer cannot know *which* votes passed the validity tests. Duplicate votes can be removed by self-comparison of the votes *before* re-mixing and re-encrypting and performing the comaprison to the official credentil list, so that nobody knows how many votes any authorized voter cast.

Finally, although a voter could provide the plaintext form of his credential σ to anybody, that would not help them to identify a vote containing an encryption of it, because no individual knows the decryption key, and the encryption method is ElGamal randomized, and there is no way for them to verify that the "credential" is not merely some random bits.

JCJ is usable with any vote-method in which votes are *anonymizable*, i.e. in which it is usaully infeasible or useless to produce a *unique* vote. It is *not* necessary that the votes be *additive* and indeed anonymizable-vote systems are a strict superset of additive ones since we can bit-split votes in any additive scheme into "single-bit" votes. The JCJ scheme is unuseable with voting systems such as reweighted range voting [150] in which it is feasible for voters to uniquify their votes.

The **main defect** of the original JCJ scheme was that processing V votes by N voters required $O(NV)$ steps to perform all the cross checks, i.e. at least N^2 steps. With 10^8 voters, the huge number of proofs of failed plaintext-equality-tests that need to be provided to all verifiers to justify discarding the bogus votes would be absolutely unacceptably huge. Fortunaely Smith [152] was able to show how to reformulate the JCJ scheme to allow the cross checks to be done via "hashing" in only $\approx 100(N+V)$ modexp steps.

7.5 Voting via secret sharing and secure general multiparty computations

At first, the election desiderata of §2 seemed irreconcilable. But then we saw that, at least for many popular vote-combining methods, both mixnets and homomorphic encryption could do it.

Both these approaches relied (although in the homomorphic case only in a minor way at the very end) on having distinct, independent, and mutually mistrustful entities performing different parts of the computation, as opposed to just having the election authority do everything. That was not a coincidence:

Theorem 5 (Impossibility). *The election desiderata of §2 are irreconcilable if the election is performed by a single entity which does everything in polynomial time, and if the vote combining method is sufficiently general.*

Proof: One "vote combining method" once suggested by Gibbard [75] (it is one of the two "strategyproof" nondeterministic voting methods) is simply to pick a random voter i and do whatever he wants (the "random dictator" method)! In order to be able to carry out a maximally general vote combining-method, therefore, the election authority would have to be able to *know* the vote of voter i, for each and every i, after at most a polynomial amount of thinking. But that contradicts the desire for ballot secrecy. Q.E.D.

The way to avoid this impossibility theorem is seen once we recognize that "single" is its key word.

Considering theorem 4, we see that it *is* possible for a set of *several* sharers to, in combination, perform *any* polynomial-time vote combining method on votes which are "shared secrets." Each voter would initially publically zero-knowledge prove the validity of his vote, and then would share his vote among the sharers using the methods of §4.11. Those sharers then would not know any votes (except if $> T$ of them colluded, but we assume they are sufficiently mistrustful that that will not happen). But they could by using theorem 4 perform any polynomial-time vote combining procedure to deduce the election result (in shared-secret form). This computation would be performed in a way that produced a zero knowledge proof of its correctness. Finally, the sharers could cooperatively determine what that election result was.

The problem with this approach is its immense communication and computational needs. Let us suppose there are 3 sharers, since that would seem to be the minimum arguably-acceptable number. Each logical AND-gate operation in the vote-combining algorithm (regarded as split into the individual bit-operations it performs) is simulated with the aid of 51 modular exponentiations. Assuming we are holding a reweighted range voting election among 10^8 voters, the total number of logic-gate operations needed would be $\approx 10^{13}$. So the total amount of computing required to make this all work would be equivalent to, say, 5×10^{14} modular exponentiations, each of which (using `Zmodexp`) takes 4.66msec on a Pentium-II/350MHz. The total amount of Pentium-time required for all that computing, then, would be 75,000 years. This could be achieved in 1 computing-day if 27 million Pentiums were assigned to the task (if massive parallelism were possible, which for fully general vote-combination methods is doubtful, but seems plausible for reweighted range voting), at a hardware cost (assuming \$1000 per Pentium) of $\$27 \times 10^9$, or \$270 per voter. These computers would require their own 2-gigawatt power plant. This cost seems too high to be justifiable, but certainly is not *impossible*.

But now consider the communication requirements. The 3 sharers would necessarily have to be in physically well-isolated locations. Each bit operation requires the communication of somewhere around 1kbit of information from each sharer to the others. The total amount of information transmitted, then, would be 3×10^{16} bits. Assuming 1 Gbit/sec communications links, this would require 116 days to transmit over 3 lines. This is outrageous. But now if we also consider the communication and computational requirements on the *verifiers* (who are supposed to be small groups without enormous fininicial and computational resources) then it becomes completely unacceptable.

So while secure general multiparty computations solves the problem in principle, with current computer and communication speeds and costs this solution is not acceptable.

8 We trade fire with electronic voting opponent Rebecca Mercuri

Rebecca Mercuri is the author of a Computer Science PhD thesis on voting, a professional voting consultant, founder of her own software company, often appears in the popular press, and also has years of experience as a poll-worker in elections. She also advocates the "Mercuri method," (which others have called the "TruVote" system and attributed to Athan Gibbs), of having electronic voting machines produce paper vote-records as follows:

1. Voter votes.
2. Machine prints out a description of that vote, which voter views through glass.
3. Voter, if satisfied, indicates approval by pushing a switch.
4. Votes are recorded and paper ballot drops into a box to be stored for possible later use in a recount.

Mercuri begins her "statement on electronic voting" [111] with words to warm the heart of any Luddite:

I am adamantly opposed to the use of any fully electronic or Internet-based systems for use in anonymous balloting and vote tabulation applications.

She then lists many reasons for this stance "based on a decade of research." Unfortunately, it will soon become clear that Mercuri knows little about the cryptographic voting methods we have discussed here. Consequently, some of her statements are simply false. However, there is much to be learned by considering them, and that is what we shall do here, in the form of an artificial dialogue between Mercuri and ourselves. Our statements pertain to the homomorphic system of §7.3; hers are extracted from [111] (nearly the entirety of [111] is duplicated below in pieces).

Mercuri [111]: Fully electronic systems do not provide any way that the voter can truly verify that the ballot cast corresponds to that being recorded, transmitted, or tabulated. Any programmer can write code that displays one thing on a screen, records something else, and prints yet another result. There is no known way to ensure that this is not happening inside of a voting system.

Reply 1: Our model assumes that the voter is casting votes from his *own* machine, whose software is written by somebody he approves of (there would be many competing vendors). In our systems the voter *is* capable of verifying that his vote *was* transmitted and recorded (in the system of §7.3 it is posted on a public bulletin board under than voter's name in signed and encrypted form). The voter is capable of verifying it is bitwise identical with the vote he produced. Anyone is capable of verifying that that vote could not have been produced by anybody other than that voter, and it could not have been altered in even a single bit. Anyone is capable of verifying – and these verifications can be done entirely on their own machines, using software written by anybody they please – (and in fact these verifications *will* be performed by many) that the "tabulation" included exactly all votes posted on the bulletin board.

So Mercuri is wrong. But she is correct that "spoofing" software or hardware acquired by gullible voters could be a problem. Spoofed smart cards that always voted Republican could in principle be manufactured and distributed to voters. They would be detectable by using them in 1-voter test "elections"; such test elections could be provided as a service by many independent groups. (Those test groups again could be lying, but there could be many of them, presumably not all lying.)

Mercuri: Electronic balloting systems without individual print-outs for examination by the voters, do not provide an independent audit trail (despite manufacturer claims to the contrary). As all voting systems (especially electronic) are prone to error, the ability to also perform a manual hand-count of the ballots is essential.

Reply 2: We have demonstrated how to produce paper printouts (of the exact same information posted on the "bulletin board"). These would be capable of being scanned in and would allow a recount, with the same verifiability properties, to be done at any time, even if the entire internet was destroyed. It would *not matter* if the election authority's vote-counting software or hardware was erroneous, cheating, or compromised – i.e. it would still be impossible for a wrong election result to be computed without detection. For this, the *only* thing that matters is that *somebody* somewhere has valid verification software with which to check the election authority. Such verification programs could be written by many independent programmers and run on many independent machines by different verification groups in different countries, etc.

Mercuri: No electronic voting system has been certified to even the lowest level of the U.S. government or international computer security standards (such as the ISO Common Criteria or its predecessor, TCSEC/ITSEC), nor has any been required to comply with such. No commercially available e-voting system has been verified as secure.

Reply 3: Most or all commercially available voting systems are indeed, as of 2004, crap. (See §9.) However, we repeat that the procedures we have described here achieve a *vastly greater* degree of security than any previous election scheme ever used. We repeat that nobody, ever, in the entire history of humanity, despite great effort by many very talented people, has ever solved a 500-digit hard integer factoring problem. Ever. Presently known methods would not succeed in doing so even if all the computers in the entire world were devoted to the task for 100,000 years. We have shown how to link the job of defeating our voting system's mathematical guarantees to the solution of harder problems than that. Meanwhile every voting system Mercuri likes has been compromised many times by unskilled adversaries. In short, it is quite likely that our voting system's mathematical guarantees will never be broken; the only fruitful avenue of attack is therefore on something else (e.g., physically preventing voters from producing the input to our system, or physically preventing our system from being used at all, or falsifying the assumptions about the world that the mathematics rests on, all are *far* easier ways to attack it than trying to break the system itself).

Mercuri: There are no required standards for voting displays, so computer ballots can be constructed to be as confusing (or more) than the butterfly used in Florida, giving advantage to some candidates over others.

Reply 4: This issue has nothing to do with whether the election is electronic or not. We agree with Mercuri that this lack of standardization is an easily-repaired outrage. In the event electronic voting became prevalent, such standardization would be more, not less, likely to happen.

Mercuri: Electronic balloting and tabulation makes the tasks performed by poll workers, challengers, and election officials purely procedural, and removes any opportunity to perform bipartisan checks.

Reply 5: The part about "removing opportunity for checking" is totally false. (The part about "procedural" is true, but that is not a problem, but rather a desirable goal.)

Mercuri: Any computerized election process is thus entrusted to the small group of individuals who program, construct and maintain the machines.

Reply 6: On the contrary: there can be an arbitrarily large number of independent verifying groups. Mercuri here has the wrong mindset – she imagines that there is just one election authority, running software which the rest of us have to trust. The right mindset is: that software has to provide *proofs* of success, checkable by anyone anywhere using independent software and hardware. Any false proofs can and will be detected.

Mercuri: Although convicted felons and foreign citizens are prohibited from voting in U.S. elections (in many states), there are no such laws regarding voting system manufacturers, programmers and administrative personnel. Felons and foreigners can (and do!) work at and even own some of the voting machine companies providing equipment to U.S. municipalities.

Reply 7: Interesting. (And true, see §9.) But the systems of the sort we are discussing are in no way hurt by internationalization or felons...

Mercuri: Encryption provides no assurance of privacy or accuracy of ballots cast.

Reply 8: Completely false. It totally protects privacy. "Accuracy" (by which I assume she here means "legitimate formatting") is ensured by the fact that each voter must provide a zero knowledge proof of his vote's legitimate formatting, accompanying that vote. Thus our system would in fact provide *far* superior voter privacy and formatting guarantees versus old-style voting.

Mercuri: Cryptographic systems, even strong ones, can be cracked or hacked, thus leaving the ballot contents along with the identity of the voter open to perusal.

Reply 9: On the contrary, we have linked "cracking" our systems to solving large discrete logarithm problems. Nobody has ever succeeded in doing that, so until and unless that day comes, cracking is not feasible and ballot contents are unperusable. "Hacking" is not relevant to the issue of whether the election can be verified, unless the hackers also manage to hack the independent systems of every verification group worldwide.

Mercuri: One of the nation's top cryptographers, Bruce Schneier, has recently expressed his concerns on this matter, and has recommended that no computer voting system be adopted unless it also provides a physical paper ballot perused by the voter and used for recount and verification.

Reply 10: Such recountable and verifiable paper ballots are produced by the system we have discussed here, and they are far *more* verifiable than any old-style voting system, since they are unforgeable.

Mercuri: Internet voting (whether at polling places or off-site) provides avenues of system attack to the entire planet. If a major software manufacturer in the USA could not protect their own company from an Internet attack, one must understand that voting systems (created by this firm or others) will be no better (and probably worse) in terms of vulnerability.

Reply 11: True to this extent: "denial of service" attacks would be possible, i.e. preventing the election from being held, but undetectable election-faking attacks would be impossible. The election system we propose is capable of of working without needing the internet while the votes are collected – except for one thing: if voters want to see their votes instantly posted to a world-viewable electronic "bulletin board," then this is not possible if communications are shut down, and such postings will be delayed until the communications are restored. (Even then postings to a *local* sub-bulletin board would still be possible.) Even if so, the situation still would be far superior to pre-electronic systems.

Mercuri: Off-site Internet voting creates unresolvable problems with authentication, leading to possible loss of voter privacy, vote-selling, and coercion. Furthermore this form of voting does not provide equal access for convenient balloting by all citizens, especially the poor, those in rural areas not well served by Internet service providers, the elderly, and certain disabled populations... off-site Internet voting systems should not be used for any government election.

Reply 12: She's right. More precisely, authentication is not "unresolvable" – digital signatures work if voters are capable of and willing to keep their keys private – the problem is voters who want to sell votes. Vote selling and coercion would be possible in most schemes if voters were voting in a non-private setting.[78] So we recommend that voters be required to vote in private voting booths rather than from arbitrary internet sites. Disabled people have always had, and if off-site voting is forbidden will always have, lesser access. Our system requires each voter to own their own personal "digital assistant" (smart card?) to use in the voting booth. If these were available for free, then poor people would not be disadvantaged. The possibility of voters selling their cards would have to be defeated by making each card only useable by their owner (e.g. because of photo and finger- or toe-print indelibly imprinted on the card; fingerprints could be taken at the polling place to try to prevent anyone voting twice with two different cards, one with forged prints) and this separability of voters from their digital assistants is among the weakest aspects of our proposed system – the mathematical model regards voters and their digital assistants as the same entity.

Mercuri: It is a known fact that the computer industry does not have the capability, at present, to assure a safe, reliable election using only electronic devices.

Reply 13: Our scheme is feasible with today's technology. On a cost-per-voter basis, it is not even expensive (well under 1 dollar per voter to adopt, excluding the cost of the "smart cards").

Mercuri: Investigation of vendor claims (such as those performed by New York City on DRE products), and failures of performance in actual elections, have demonstrated the existence of major flaws.

Reply 14: Flawed voting machines would be *far more*, not less, detectable under our system, since each voter could immediately verify the fact that his vote was (or was not!) posted on the world-viewable bulletin board. Most flawed voting machines would be detected that same day. Voting machines failing to obey the protocol for communicating with the voter's digital assistant would be detected immediately. Any voters whose votes did not get posted, would be fully capable of trying again to vote, until eventually they succeeded. There would be no penalty for multiple voting (although only the most recent vote would actually be used).

I have corresponded with Mercuri over the years and attempted to point these things out to her, but she would not modify her "statement" [111]. However, she did try to indicate that her "statement" had been intended to be directed toward currently commercially available voting machines, not toward theoretical developments. They are far more acceptable if viewed in that light.

9 Examples of real world voting frauds, errors, deceptions, suspicious events, and stupidities

The following stories have mainly been extracted from [14] [32] [33] [36] [68] [81] [84] [90] [103] [113] [122] [104] [158] to illustrate the variety of known kinds of election fraud and manipulation techniques, as well as unintentional errors.

9.1 Voter registration and eligibility

Closeup on Duval County, Florida:

The *Washington Post* [14] found that Duval's rejected registrations to vote in the 2004 election were 35% black, although only 20% of *accepted* registrations were by blacks. Some registrations were not rejected but instead merely "flagged" as incomplete. There were nearly 3 times the number of flagged Democratic registrations as Republican. Broken down by race, no group had more flagged registrations than blacks.

Secretary of State Glenda E. Hood (appointed by Gov. Jeb Bush) ruled that for registrations to be deemed complete, new voters must not only sign an oath attesting to their citizenship, but also check a box that states the same. Unlike many counties, which have chosen to ignore that directive, Duval County chose to enforce it. (There are also other boxes that "must" be checked.)

Duval County used punched card voting machines. According to Harris [84]: "One way to rig a punch card system is to consolidate ballot-counting in one area so that precincts are mish-mashed together; Then, the scoundrel team picks someone to quietly add punches to the votes... The double-punched cards become 'overvotes' and are thrown out." In 2000 in Duval County 21,942 overvoted punch cards appeared (equivalent to 1 overvoted card per 10 cards) with most of the overvoted cards coming from just 5, all heavily black, out of the 268 precincts in Duval. No one was allowed to look at them. Duval had the highest ballot rejection and overvote rates among all Florida counties.

A statistical analysis of the Duval precinct totals was made by the statistics consulting firm `netrinsics.com` [53]. One hypothesis that had been advanced in the press [23] to "explain" this without the existence of any skullduggery, was that many hastily mobilized democratic first-time voters had been confused by instructions to "punch every card." However, the Netrinsics statistical analysis nixed that hypothesis. Netrinsics' point was that only the *presidential* ballot was spread

[78] But see lesson 6 of §11.1.

over more than 1 card (because there were 10 presidential candidates).

> One would expect, in consequence, that the missing vote rate for non-presidential races in Duval County would be similar to comparable counties. However, by comparing these races with the plots in figure 3, one notices that Duval County missing vote rates in the senatorial, and treasurer races also show an anomolous trend upward with increased Democratic participation... This hypothesis does not explain how the missing votes in the presidential race are proportionate [to the Gore vote] regardless of precinct party composition, where the education race shows significant variation depending on party balance. The hypothesis also does not explain the very large number of missing votes for the senatorial and treasurer races found in Democratic precincts in Duval County, but not other counties.
>
> Approximately 20,000 presidential votes have gone missing in Duval County (above and beyond what would normally be expected).

Netrinsics also compared Duval to Lee County [117], noting that both employ the same sort of centrally tabulated punch card voting machines and have about the same population. But Lee County had 1/4 of Duval's rate of invalid punch-card ballots. Unlike Lee, which is a fairly homogenous county without any extremes, Duval has a large number of predominantly Republican precincts, with a smaller number of extremely Democratic precincts. The highest rate (22%) of invalid ballots in Duval occurred in those precincts voting $\leq$ 10% for Bush, with the invalid-ballots rate showing an excellent (negative slope) linear fit to the Bush-voting rate countywide (the precincts voting $\geq$ 80% for Bush had 3% invalid ballots). See fig. 11.1. Netrinsics concluded:

> One explanation that fits all the evidence is that some aspect of the tabulating process in Duval County is biased against Democratic votes. Another explanation is that 20% of all Democratic voters throughout Duval County, and only in Duval County, suffer from an unexplained voting impediment which is not evident in other counties around Florida, such as Lee County.

Some counties provide "early voting" locations. Orange County, which has approximately the same number of registered voters as Duval, chose to open nine for the 2004 elections, but Duval only one – even though Jacksonville is geographically the largest U.S. city, covering 840 square miles. It was miles from most of the black precincts.

Florida 2004: 58,000 absentee ballots which were supposed to have been mailed out on Oct. 7-8 mysteriously vanished in Broward County (the Florida county which had voted against Bush by the largest margin in 2000; Gore got 67% of Broward's votes) with the county blaming the post office but the post office blaming the county [8]. Assuming all those voters have been disenfranchised by this development and assuming the same voting ratios occur in 2004, this alone accounted for 19,000 extra votes worth of Bush-Kerry margin. Students at Florida State and Florida A&M universities, some of whom signed petitions to legalize medical marijuana or impose stiffer penalties for child molesters, unknowingly had their party registration switched to Republican and their addresses changed [13]. (The address change invalidated their votes.)

Ohio 2004: Voter registrations were being rejected if they were printed on an insufficiently heavy grade of paper. J.Kenneth Blackwell, the Ohio Secretary of State (in charge of elections) was, in a reprise of Florida 2000, simultaneously the Bush-Cheney state campaign co-chair. He wanted "white, uncoated paper of not less than 80 lb. text weight." (Eventually Blackwell relented on this policy, but only after using it for thousands of registration rejections. According to the League of Women Voters, this was the only policy of its kind in the country.)

Ohio 2004: The Ohio Republican Party performed "caging" [147]. That is, it sent registered letters to newly registered voters in minority and urban areas, then tried to challenge 35,000 people who did not sign for the letters (or if the mail otherwise came back as undeliverable – including voters who were homeless, serving abroad, or simply did not want to sign a delivery from the Republican Party). Notice of the Republican-intended challenge hearing was sent to the 35,000 voters far too late to be of any use to those challenged, indeed some received it after the election was over. These moves were eventually struck down by 3 separate court decisions.

Michigan 200?: Michigan found 1 million duplicate registrations throughout the state when it created a unified statewide registration system.

Israel 2004: Non-Jews have always been extremely unrepresented both in the Knesset and in the voting population.

USA, 1880-1950: "Jim Crow" refers to policies systematically preventing blacks from voting, as well as a pattern of other kinds of segregationist policies and laws. It operated throughout the country, especially in Southern states. Various pseudo-legal disenfranshisement measures were employed, including tests of "literacy" administered at polling places which conveniently varied from extremely easy to extremely difficult, "poll taxes," and "grandfather clauses" and "good conduct clauses" which conveniently exempted whites from having to pass these hurdles. (One common "literacy test" was to require the black would-be voter to recite the entire U.S. Constitution and Declaration of Independence from memory.) Such laws proliferated after the Supreme Court 1896 ruling in *Plessy v. Ferguson* that segregation was legal and in *Williams v. Mississippi* in 1898 that the following Mississippi literacy-test stipulation in its constitution, was legal: "On and after the first day of January, A. D. 1892, every elector shall, in addition to the foregoing qualifications, be able to read any section of the constitution of this state; or he shall be able to understand the same when read to him, or give a reasonable interpretation thereof." (Power was granted to registrars – all white political appointees – to interpret the test.)

James Kimble Vardaman (later Governor of Miss.), boasted of the constitutional convention that created this: "There is no use to equivocate or lie about the matter. Mississippi's constitutional convention was held for no other purpose than to eliminate the nigger from politics; not the ignorant – but the nigger."

How effective were these measures? Over 130,000 blacks were registered to vote in Louisiana in 1896, but there were only 1342 on the rolls in 1904.

But probably the most effective barrier to black political power was the Democratic party primary. Since the Democratic Party dominated the South, its candidates always won; primaries were the real election. Beginning in the 1890s Democrats were able to bar blacks from voting in the primary on the pretext that the party was a private club not subject to federal anti-discrimination laws.

California 1999: CBS's *60 Minutes* found people in California using mail-in forms to register pets and fictitious people, then obtaining absentee ballots in their names. This included an elephant at the San Diego Zoo (Republican?).

Meanwhile in St. Louis Missouri, it was discovered that voter rolls included 13,000 more names than the U.S. Census listed as the total number of adults in the city.

Pittsburgh Pennsylvania 2004: Fliers were handed out at a Pittsburgh area mall, and mailed to an unknown number of homes. The flier, on bogus but official-looking stationary with a county letterhead, told voters that "due to immense voter turnout expected on Tuesday," the election had been extended: Republicans should vote Tuesday, Nov. 2, it said – and Democrats on Wednesday [13].

Nevada 2004: [103] Private voter-registration company Voters Outreach of America (paid $488,000 by the Republican National Committee) employed up to 300 part-time workers to collect hundreds of voter registrations per day in Las Vegas, Nevada. But, its former employees said, Voters Outreach only wanted Republican registrations. Two told George Knapp of KLAS TV they personally witnessed company supervisors rip up registration forms signed by Democrats. Employee Eric Russell managed grab some shredded forms, all signed by Democrats, from the garbage and KLAS TV took them to the Clark County Election Department and confirmed that they had not been filed with the county as required by law. Russell also left some with the FBI. Many who thought they would be able to vote on Election Day were therefore sadly mistaken.

Voters Outreach then vanished from Nevada, leaving their landlord complaining about nonpayment of rent.

Voters Outreach also operated in Portland, Oregon under the name "America Votes," which is in fact the name of a Democratic organization. Employees in Las Vegas say they too were told that the name of the company was America Votes. "They confused us with the name. They told us one thing and told the temp force something else. They told us America Votes," Russell said.

USA versus other countries, 2000: In 33 out of the 50 US states, voters need not present identification either to register or to vote. This is quite odd considering that a Rasmussen poll found that 82% of Americans believe that "people should be required to show a driver's license or some other form of photo ID before they are allowed to vote."

Mexico and many other countries have far more secure election systems than the USA's. Citizens must provide a photo, a signature, and a thumbprint to register. The voter card includes a picture with a hologram covering it, a magnetic strip and a serial number. To vote, you must present the card and be certified by a thumbprint scanner. This system was instrumental in allowing the 2000 election of Vicente Fox, the first opposition party president in 70 years.

9.2 Vote collecting

Oregon 2000 [84]: "Scoundrels stood on street corners with official-looking boxes to 'collect' absentee ballots."

9.3 The story of the Ukraine election in 2004

Reporters witnessed groups of 30 thugs in masks invading polling places and beating voters and officials, and the destruction of ballot papers en masse by pouring acid into ballot boxes [123]; there was also "carousel" voting, in which busloads of Yanukovych supporters simply drove from one polling station to another casting multiple false absentee ballots.

Observers from OSCE (Organisation for Security and Co-operation in Europe) saw voters given pens filled with disappearing ink, leaving ballots unmarked and invalid. Freedom House [105] reported "massive voter fraud" while ENEMO (European Network of Election Monitoring) "identified a systematic pattern of violations that seem to have been planned to influence the elections... the blatant and carefully targeted violations ensured the election does not reflect the will of the Ukrainian people."

Monitors confirmed: intimidation and expulsion of election monitors, ballot stuffing, multiple voting, and government pressure on voters.

Opposition presidential candidate Viktor Yushchenko was denied media access via media temnyks (theme directives from the government guiding television news presentation), disruptions of public rallies, official harassment, beatings and arrests of hundreds (including journalists), and searches of civic group offices. Yushchenko was also poisoned with dioxin. Despite this most Ukrainian voters appear to have voted for Yushchenko, according to two separate exit polls (one showing a 4% and the other an 11% margin).[79] But the official results released by Ukraine's Central Election Commission on 23 Nov. showed incumbent Prime Minister Viktor Yanukovych had won by 3%. These included counts showing that most districts of Donestk (in East Ukraine) gave Yanukovych 97% of their votes, with 98.5% turnout, among about 1 million extra voters beyond those registered for the original election (this, crucial, election was a runoff).

In public institutions, such as prisons, hospitals, and psychiatric institutions, Yanukovich won by massive margins that were often contrary to the prevailing trends in their localities. (OSCE mentioned evidence students, government employees and private sector workers were forced by their deans and supervisors to vote for one candidate over another.)

[79] Why the large discrepancy between the two polls? Both polls have been criticized as flawed: One relied on face-to-face interviews, so that the pollees had to worry they might face retribution; the other inadequately sampled precincts in Yanukovych strongholds.

The Ukrainian Supreme Court annulled the results and ordered a repeat of the Yuschenko-Yanukovych runoff. The re-run was held on December 26. Observers reported a much fairer vote, and Yushchenko won by 52%-44% over Yanukovych.

9.4 The USA 2004 presidential election, with focus on Ohio

When Bush beat Kerry in 2004 to win his second US presidency, the wide perception was that, unlike the ultra-close 2000 election, this one was not close and therefore could not have been tipped by electoral fraud and fraud-like developments.

Examining the situation more closely reveals disturbing facts.[80]

We shall consider the hypothesis that extensive fraud and fraud-like events aided the Bush side in the 2004 election. From that hypothesis would follow **two predictions:**

1. We would expect most states to have systematically larger pro-Bush official vote totals than those predicted by unofficial exit polls, with the largest discrepancies occurring in the 11 consensus "battleground" states,[81] because the same amount of fraud in a battleground state yields a greater impact on the election. Nevertheless because the battleground states *are* battlegrounds and are heavily scrutinized by the opposition, we would not expect *vastly* greater fraud would be achieveable within them, in general.
2. We also would expect that a larger amount of the fraud would occur in pro-Bush "stronghold" precincts, since that is where it would be easiest to accomplish without detection.

Consider the CNN (Cable News Network) exit polls, conducted in all states nationwide plus DC (except Oregon) by the Edison-Mitofsky polling organization. As table 9.1 shows, [82] *the official counts were more in Bush's favor than the exit polls*, in 10 out of 11 battleground states, with the 11th staying the same. In fact, the exit polls predicted a Kerry victory, while the official counts yielded a Bush victory. If CNN's exit polls were unbiased, then the a priori probability of this one-sided discrepancy would be 1/1024 – the same as the probability that if 11 coin flips were performed and then the first 10 were examined (the 11th remaining undetermined) those 10 all would yield heads.

State	sample	$B-K_{\text{poll}}$	$B-K_{\text{official}}$	discrep	WPE_{EM}
CO	2515	+1.8%	+4.7%	+2.9	−6.1
FL*	2846	+0.1%	+5.0%	+4.9	−7.6
IA	2502	−1.3%	+0.7%	+2.0	−3.0
MI	2452	−5.0%	−3.4%	+1.6	−6.3
MN	2178	−9.0%	−3.5%	+5.5	−9.3
NV	2116	−1.3%	+2.6%	+3.9	−10.1
NH	1849	−10.8%	−1.4%	+9.4	−13.6
NM	1951	−2.6%	+0.8%	+3.4	−7.8
OH*	1963	−4.2%	+2.1%	+6.3	−10.9
PA*	1930	−8.7%	−2.5%	+6.2	−8.8
WI	2223	−0.4%	−0.4%	0	−4.7
avg	2229	−3.8%	+0.6%	+4.19%	−8.02

Figure 9.1. CNN's exit poll results at 12:21-12:24am EST on 3 November 2004 (recorded by Jonathan Simon – as well as by others independently in at least 17 cases) versus official results in the 11 battleground states for 2004 Bush-vs-Kerry presidential election. CNN gave four results, rounded to the nearest integer percent, for both male and female voters and for both Bush and Kerry in each state polled; and also reported the total respondent count and the male and female percentages rounded to the nearest integer percent. Both genders have been combined here and reported accurate to ±0.05%, and with the resulting Kerry number subtracted from the Bush number. The three "critical" battleground states (which the candidates visited the most and spent the most money in) are starred (*). The poll results are compared with the official results from the Federal Election Commission. (The "WPE" column is from [55] and is explained later.)

After about 1am, the CNN exit poll results changed. CNN's final exit poll figures are not reported here because they were "adjusted" by incorporation of official totals and therefore are useless for our purposes (a little-known and little-reported fact[83]). ▲

[80] Important sources for this section include: Official Ohio vote totals were from the Ohio Secretary of State's office. Figures about uncounted votes in Ohio counties are from Dr. Richard Hays Phillips, who testified before the Ohio Supreme Court about them. Some ideas are from Steven F. Freeman [65] but all my calculations are independent of his and I do not endorse much of what he says. The following errors and deceptive statements appear in [65]: Freeman makes exit polls appear typically to be extremely accurate by tabulating German exit poll data. (His point is that it is astonishing that the exit polls in the USA in 2004 were so far off.) He leaves unmentioned two exit polls in the Ukraine 2004 and all the Edison-Mitofsky polls for the last five US presidential elections (I know he was aware of both since he mentions them in his other writings) – all 7 of which were far less accurate than his German data. Freeman also gives a version of our table 9.1 but in which 7 of his 11 "official state election totals" are in error. In every single case, Freeman's error just happened to be of the right sign to exaggerate his case. To use a similar statistical analysis to Freeman's own, the probability that 14 independent figures all should be deceptive or off in the right direction to exaggerate his case, would be, in the absence of intention, $2^{-14} = 1/16384$.

[81] There were exactly 11 states mentioned as "battlegrounds" by at least 2 of these 3 lists: Zogby, MSNBC, and Washington Post, namely CO, FL, IA, MI, MN, NV, NH, NM, OH, PA, WI. Also OR was sometimes mentioned as a battleground but cannot be studied since it had voting by mail and hence exit polls were inapplicable to it. These 12 were the only states with margins below 7%.

[82] Note the "+" signs in the "discrep" column.

[83] It was widely reported [166] that the Caltech/MIT voting project had "debunked" this "exit poll discrepancy." However, that report [29] was based on the "final" CNN exit poll numbers – which had already been adjusted to conform with the official results! – and therefore their finding of no great discrepancy is meaningless. This report also attacked the earlier non-final CNN poll data as "too female" as a result of time-of-day bias. (Females in the absence of males would have elected Kerry.) These data are from CNN's web page at 12:21am on election night. If the CNN data (which came broken down by gender) were renormalized to 52/48 female/male ratio (commonly accepted), then the nationwide pro-Bush margin-shift would have been reduced by an additive amount of 0.4%, and with 50-50 sex ratio by 0.8% (according to Simon). These changes are tiny in comparison to the magnitude 4.19% here (or 6.5% as later stated by [55]) for that shift. Without trusting Simon, you still can easily see the "too-female" effect had to be small since females were only overrepresented by order 2% in the early day, so the correction to the shift, if it were due to gender sample bias, had to be only of order 4% *multiplicatively*; the Caltech/MIT report should have been able to realize this but never performed this simple calculation.

The 35 "safe" states also showed a discrepancy between exit polls and official results – but a smaller one, amounting to an average $\approx +2.8\%$ margin-shift toward Bush. Meanwhile the 11 "battleground" states in table 9.1 showed an average shift of $+4.19\%$. This is despite the fact that the exit poll sample sizes were nearly twice as large in battleground states (2200) as in the other states (1150), which theoretically should have resulted in *smaller* polling errors.

The one-sided error nature of all this was completely compatible with prediction #1.

Edison-Mitofsky released a 77-page study of their exit poll discrepancy [55]. Its summary stated:

> The exit poll estimates in the 2004 general election overstated John Kerry nationally and in many states. There were 26 states [more than half] in which the estimates produced by the exit poll data overstated the vote for John Kerry by more than one standard error, and there were four states in which the exit poll estimates overstated the vote for George W. Bush by more than one standard error...
>
> The inaccuracies in the exit poll estimates were not due to the sample selection of the polling locations.
>
> We have not discovered any systematic problem in how the exit poll data were collected and processed.
>
> Exit polls do not support the allegations of fraud due to rigging of voting equipment... [We] found no systematic differences for precincts using touch screen and optical scan voting equipment. [Unfortunately they *did* find a very large systematic difference between old-fashioned paper ballots and every other kind of voting equipment ([55] p40).]

Edison-Mitofsky noted that their WPE (*w*ithin *p*recinct *e*rror) averaged -6.5% (their negative sign means the polls overestimated the Kerry$-$Bush difference) averaged over all 1460 precincts for which they had official results, which was "the largest WPE that we have observed on a national level in the last five presidential elections." The WPE was positive in only 6 states and negative in the remaining 44.

Edison-Mitofsky simply ignored the possibilities that either the official election results were fraudulent, that Kerry voters who thought they had voted were in fact less likely to have their votes recorded and counted, or that their own polls were fraudulent or somehow intentionally skewed by some external agency e.g. by bribing or contributing pollsters. They *concluded* that, due to some unknown reason, "Kerry voters were more likely to participate in the exit polls than Bush voters."

This Edison-Mitofsky "explanation" could indeed explain the overall nationwide pro-Bush shift. However,

1. It cannot explain the fact (with which they agree [55] p.42) that there was a larger pro-Bush shift in the battleground than in the other states: according to [55] their average WPE was -6.5% nationwide, -8.02% in the 11 battleground states, and -9.10% in the three critical battleground states.
2. It cannot explain the fact that in the 2000 Bush-Gore election, Edison-Mitofsky polls ([55] p34) had unbiased errors, with 394 precincts with WPE< -5 and 315 with $WPE > +5$ and 374 with $-5 <$WPE$< +5$, while in 2004 there were drastically biased errors with 767 precincts with WPE< -5 and 352 with $WPE > +5$ and 341 with $-5 <$WPE$< +5$.
3. On the top of page 37 of the Edison-Mitofsky report is data indicating that in more pro-Bush precincts, a higher percentage of people agreed to participate in the polls than in pro-Kerry precincts. (E.g. in precincts voting $\geq 80\%$ for Bush, 56% of those asked agreed to be in the exit poll. In precincts voting $\leq 20\%$ for Bush, only 53% agreed.) This indicates a sampling bias *opposite* in sign and with about half the magnitude required to make the Edison-Mitofsky "explanation" work. (On the other hand, this does support the notion that sampling biases *can* occur of roughly the right magnitude to explain a discrepancy this large.)

Next, let us consider prediction #2. In their 40 Bush "stronghold" precincts, i.e. those that voted $< 20\%$ for Kerry, Edison-Mitofsky ([55] bottom of p36) found a huge mean WPE of -10.0%. On the other hand, in the 90 Kerry strongholds ($\geq 80\%$ Kerry votes) the mean WPE was $+0.3\%$, while in the remaining 1110 (non-stronghold) precincts the WPE averaged -7.3%. This is despite the fact that theoretically (without fraud) there simply is "less room" for polls to overstate Bush in high-Bush areas, so one would theoretically expect the WPE to be *less* negative in the Bush strongholds. So this finding again is entirely compatible with our fraud hypothesis.

Let us now take an in-depth look at the crucial state of Ohio, which, if it had chosen Kerry, would have elected him President.

The CNN exit poll screenshot at 1:05am showed 1963 respondents:

```
females (53%):  Bush 47%   Kerry 53%
males   (47%):  Bush 49%   Kerry 51%
```

and after 1:41am showed 2020 respondents:

```
females (53%):  Bush 50%   Kerry 50%
males   (47%):  Bush 52%   Kerry 47%.
```

This sudden and drastic change is interpreted as caused by a "correction" made by the pollsters to incorporate the official election results, because it was almost impossible for the 57 additional respondents to cause this large a shift in the results.[84] The ultimate official Ohio totals were 50.82% Bush and 48.70% Kerry, and among Bush and Kerry voters *only*, they were 51.06% Bush and 48.94% Kerry.

Under pessimal-for-Bush integer-rounding assumptions maximally intended to hurt our fraud hypothesis, the 1963-respondent Ohio poll really was

[84] It was just barely possible if all 57 voted Bush and if all CNN's roundings-to-integers were maximally favorable for this.

```
females (52.5%):  Bush 47.5%   Kerry 52.5%
males   (47.5%):  Bush 49.5%   Kerry 50.5%
total    100.0%:  Bush 48.45%  Kerry 51.55%.
```

Now, under the model that the 1963 respondents were truly a random sample among the 5625631 official votes, and those respondents told the truth to the pollsters, and these poll numbers are unadjusted, and that all 1963 were Bush or Kerry voters, then the probability that those 1963 would have split ≤ 951 for Bush and ≥ 1012 for Kerry (as the pollsters reported) would be at most

$$\sum_{k=0}^{951} \binom{1963}{951-k} 0.5106074^{951-k} 0.4893926^{1012+k} = 0.01089. \tag{17}$$

In reality, a true random sample was not achieved because the pollsters were located at random locations, as opposed to selecting random voters. Therefore there were intervoter dependencies. Merkle & Edelman ([112] p72) claim that in practice, that effect multiplies the standard error by 1.3. So regarding the 0.01089 as a point in the tail of a normal distribution and using a 30% larger standard deviation for that distribution, this probability rises[85] to 0.0502.

We conclude that the a priori probability that the true Ohio election results would have differed this much in Bush's favor versus the exit poll results (in the absence of fraud) was at most 5%.

This is enough to make us suspicious of Ohio. Evidently the actual vote counts did not correspond to the votes that the voters thought they had cast, or the voters lied to the exit pollsters (or the pollsters themselves lied), or were a nonrepresentative sample, or this was just a 5%-chance statistical fluke.

There are numerous indications suggesting that the Ohio vote was intentionally distorted heavily in Bush's favor as part of a conspiracy orchestrated by J.Kenneth Blackwell, the Ohio Secretary of State. In a reprise of the Florida 2000 scandal, Blackwell, like his Florida analogue Katherine Harris, was simultaneously the Bush-Cheney Ohio campaign co-chair, and could therefore be expected to be maximally biased. Under Ohio election law, the members, directors and deputy directors of all boards of elections are assigned by the Secretary of State. They hold these paying jobs at his discretion regardless of whether they are Democrat or Republican.

Blackwell certified a 98.6% turnout in the Concord Southwest precinct of Ohio's Miami County, meaning all but 10 among the 689 registered voters in that precinct voted. These 679 votes contained 520 for Bush and 157 for Kerry. (In the 2000 Bush-Gore election, the same precinct had voted Bush 378, Gore 132 for 74% turnout.) The Columbus *Free Press* then found 25 registered voters in this district who said they had not voted. Unless at least 15 of them were lying, then the official tally was fraudulent [125][63].[86]

Many other distressing claimed Ohio vote anomalies and events are described in [64]. In Warren County (official total: Bush 68035, Kerry 26043) the Board of Elections claimed a "Homeland Security alert" authorized them to throw out all independent and media observers and lock the building, both during the count and a later recount. County officials said this was due to a terrorist threat "that ranked a 10 on a scale of 1 to 10" received from the FBI. But the FBI denied to a later congressional investigation that it had issued any such warning or had any information about a terror threat in Warren County – and the county officials refused to name their FBI source.

Over 106000 provisional and machine-rejected ballots were never counted nor examined in either the original or recount election in Ohio. As we shall see this was done in an extremely Bush-favoring manner *by discarding votes in pro-Kerry precincts at a higher rate than in pro-Bush precincts.*

Bush won Hamilton county (which includes Cincinnati) with 222616 votes to Kerry's 199679. There were exactly 26 precincts countywide with more than 8% of the votes left uncounted. Kerry won all 26 of them overwhelmingly, by an aggregate margin of 10 to 1.

In Cuyahoga county (containing Cleveland) the net score was Kerry 448503, Bush 221600. There were exactly 84 precincts in that county with $\ge 4.0\%$ uncounted ballots. Kerry won 82 of these, by an aggregate margin of about 10 to 1.

In Montgomery County there were exactly 588 precincts, and exactly 47 of them had more than 4.0% of the regular ballots uncounted. Kerry won every one of those 47 precincts, by an aggregate margin of 7 to 1. The countywide official vote totals, meanwhile, were 142997 for Kerry and 138371 for Bush. In these 47 precincts the rate of ballot "spoilage" was 5.16%, compared to 1.31% for the rest of the county.

In Summit County: there were exactly 71 precincts with more than 3.0% of the ballots uncounted. Kerry won 70 of these 71, by an aggregate ratio of about 4 to 1. Meanwhile the countywide totals were Kerry 156587, Bush 118558.

In Stark County there were exactly 28 precincts with $\ge 3.33\%$ of the regular ballots uncounted. Kerry won all 28 precincts, by an aggregate margin of 2.7 to 1. Meanwhile the countywide totals were Kerry 95337, Bush 92215.

In Franklin County (Bush 237253, Kerry 285801), there were exactly 146 wards. Of these 69 were won by Bush and 77 by Kerry. Of the 73 wards with < 300 registered voters per voting machine, 54 were won by Bush. The median turnout in these 73 wards was 62.33%. Of the 73 wards with ≥ 300 registered voters per voting machine, only 15 were won by Bush. The median turnout in these 73 wards was 51.99%. The Columbus *Free Press* and *Columbus Dispatch* suggested there was an intentional strategy of unequal distribution of voting machines – designed to make Kerry voters wait in long lines and to reduce Kerry turnout – overseen by Franklin County Board of Elections Director Matt Damschroder, former Executive Director of the Franklin County Republican Party. Sources told the *Free Press* that Damschroder and Blackwell

[85] In the sense that erfc(1.8)=0.0109 and erfc(1.8/1.3)=0.0502.

[86] Historically, the usual official response to small discrepancies such as this one has been to ignore them. – as is precisely what happened in this case. However, Di Franco et al [52] pointed out that altering just *one* vote per US voting machine would have been enough to swing the 2000 US presidential election. So if electronic voting machines susceptible to that kind of manipulation became prevalent, the historical record would provide considerable confidence to fraudsters that such fraud could succeed.

met with President George W. Bush in Columbus on Election Day.[87]

Waits to vote in several areas in Ohio were notorious in the 2004 election, with multihour waits, in some cases as much as 12 hours, reported in numerous media [131]. Thousands of Republican lawyers were mobilized to "challenge" voters in democrat-dominated precincts with the apparent goal of slowing down voting and increasing the length of queues.

In (Democrat-dominated) Cincinnati, 150000 voters were moved from active to inactive status for not voting in the last two federal elections within the last four years. This is not required under Ohio law, but is an *option* allowed and exercised by the Republican-dominated Hamilton County Board of Elections.

Other suggestions which accusers have raised are these: Republicans redrew the boundaries of democrat-dominated voting precincts and then voters showing up at the wrong location to vote, were disenfranchised. Deceptive letters were sent out by Damschroder's county election board notifying "felons" of termination of their voting rights, whereas in fact, some were not felons and most still had the right to vote. I do not know how true these accusations are, and mention them solely to provide the reader with an idea of the possibilities for subtle election manipulation.

Whether or not the above developments legally constitute "fraud," they are certainly deplorable and certainly heavily distorted the Ohio vote totals in Bush's favor. The total distortion caused by these methods is evidently of the same order of magnitude as both Bush's victory margin and Ohio's exit poll discrepancy. I do not know whether it was enough to swing the election.

As was widely reported, Ohio later, after funding was raised by the Green and Libertarian parties, conducted a recount of the 2004 election, confirming the first election's claim that Bush had won. Officially, it awarded 734 additional votes to Kerry and 449 additional votes to Bush. The final official Bush-over-Kerry margin was 118775 votes.

What was not so widely reported was: (This included two precincts in Montgomery county where over 25% of the voters supposedly chose not to vote for a presidential candidate.) *Only 3% of the votes were tested* in this "recount."??? And after the official termination of the "recount" on 28 December, over 92672 machine-rejected ballots remained uninspected, as did at least 14000 provisional ballots.

The tested votes were not a random sample,[88] contrary to the law under which the recounted precincts were to be selected at random. Further, many of them refused to permit observers of the recount. As was (again) not so widely reported, the Greens and Libertarians declared the recount unacceptably incomplete and unreliable and asked for another. They did not get it.

Offically Bush won 50.8% to 48.7% over Kerry in the Ohio votes counted on election night (mainly by Diebold electronic machines). But among the 147400 provisional and absentee ballots (mostly the former) counted (by hand) *after* election night, Kerry received 54.46% of the vote. That very significant discrepancy (20σ) indicates that Kerry votes were shunted into provisional ballots at a higher rate than Bush votes.

9.5 Vote buying

How much are votes worth? As of 2004, reportedly the largest amount of money spent by any candidate in a large election was $44 per vote in the successful 2000 campaign by now-NJ-Senator Jon Corzine. Corzine spent $65 million to get 1,479,988 votes (51%), beating his Republican opponent Bob Franks' 1,383,474 votes (47%; Franks only spent $5 million) as well as some independent candidates with $\approx 1\%$ each.

One might therefore imagine that vote-selling would be very rare, considering the small amount of money available versus the large criminal penalty if the seller (or buyer) gets caught. But on the other hand: a $44 payment on election day would exceed the average daily income of about 20% of all households in the USA's 100 largest cities, and criminal prosecutions for vote selling are extremely rare: "The number of people who have spent time in jail as a result of a conviction for voter fraud in the last dozen years can be counted on the fingers of one hand" [68]. So then one might instead imagine that, in the right circumstances, vote selling is common.

The advent of the internet has opened up new opportunities. In September 2002 a Kiel-based firm called Fortschritt had a web site at `cashvote.com` offering to deliver packages of 1,000 and 10,000 votes, for 6,250 and 59,000 Euros; the company claims to have acted as an intermediary in the sale of over 15,000 votes [90]. This is all despite German laws that make actual or attempted vote trading punishable by a fine or jail sentence of up to 5 years. On 1 Nov. 2002, Massachussetts Attorney General Tom Reilly filed suit to shut down `VoteAuction.com`; 1116 Massachusetts voters had registered on it to sell their votes, with the highest bid for those 1116 votes being $12,000.

Vote-pair/swap/trade schemes: Also in 2002, California ordered `voteswap2000.com` to be shut down, but a similar site `http://votepair.org/` remains in operation as of 2004. Both were set up as permit vote "pairing" where somebody in state #1 agrees to vote one way if somebody else in state #2 agrees to vote another. Non-secret postal ballots in the USA can make it possible to sell and swap votes with some degree of confidence.

Florida 1999: The *Miami Herald*'s 1999 Pulitzer Prize was for uncovering how "vote brokers" employed by candidate Xavier Suarez stole the 1998 mayoral election by tampering with 4740 absentee ballots. Many were cast by homeless people who didn't live in the city and were paid $10 apiece and shuttled to the elections office in vans. "I had no choice. I was hungry that day," Thomas Felder told the Miami Herald. "You wanted the money, you were told who to vote for." All of the absentee ballots were thrown out by a court four months later and Suarez's opponent installed as mayor.

[87] The official story, e.g, reported in the *Chicago Tribune* was that Bush had flown to Columbus that morning from his home in Crawford Texas, in order that he could join phone bank members by making a single phone call with them to a voter; then Bush went back to the airport and flew to Washington DC.

[88] This was admitted by Allen, Clermont, Cuyahoga, Hocking, Medina, and Vinton Counties, and discussions of what they claimed to have done are in the Conyers report [38].

9.6 Electronic voting machines

Georgia 1998 & 2000: Dozens of memory cartridges were "misplaced," representing tens of thousands of votes. There was no documented chain of custody during the time they were missing [84].

A software programming error caused votes for Sharon Cooper to vanish. According to news reports in the *Atlanta Constitution*, the problem was fixed by on-the-spot reprogramming. (Such reprogramming, however, is illegal.)

Diebold Inc. 2004: Diebold is one if the largest manufacturers of the voting machines used in the USA. 80% of all year-2004 votes in America were counted by only two companies: Diebold and ES&S. The vice-president of Diebold and the president of ES&S are brothers (Bob and Todd Urosevich).

Diebold's managers include at least 5 convicted felons (Cooper, Lee, Graye, Elder, and Dean) involved with the management and development of Diebold's systems. Senior Vice President Jeff Dean tops the list with twenty-three counts of felony Theft in the First Degree. According to the findings of fact in case no. 89-1-04034-1 (Washington State, King County District Court; Dean served prison time): Dean's thefts occurred over a $2\frac{1}{2}$-year period of time, there were multiple incidents, the actual monetary loss was substantially greater than typical for the offense, the crimes and their cover-up involved "a high degree of sophistication" in planning, using, and altering records in the computerized accounting system that defendant maintained for the victim, and the defendant "used his position of trust and fiduciary responsibility as a computer systems and accounting consultant for the victim to facilitate the commission of the offenses" [84]. Dean's reason for his embezzlement was that he needed the money because "he was embezzling in order to pay blackmail over a fight he was involved in, in which a person died." The other felons included a cocaine trafficker and a man who conducted fraudulent stock transactions. Diebold's CEO Walden O'Dell has done fundraising for Pres. G.W.Bush, including raising $600,000 for Vice President Dick Cheney at a single party on 30 June 2003, and noted in a fall 2003 letter that "I am committed to helping Ohio deliver its electoral votes to the president."

Diebold Corp. donated over $170,000 to the Republican party during 2000-2002 and its directors and chief officers donated $240,000, *all* to Republican candidates or party funds.

Two unencrypted copies of the `C++` source code for Diebold's AccuVote TS system were found on the Diebold web site, one dating from around 2000, and one dating from late 2002, available to anybody who wanted to download them. (This was despite the fact that Diebold and all other US voting machine manufacturers refuse to release their code for public inspection.) These codes were then inspected by several computer science professors [104]. One of their discoveries was that the Accuvote system used DES encryption to transmit votes (advertised as "world class cryptography"[89]). Every machine used the *same* secret key, and that key was "hardwired" into the source code, i.e. available for public view on the Diebold web site, i.e. not secret at all. (Also, it could be discovered by simply scanning through the memory of any Diebold machine, which would take 1 second. Supposedly, DES is tremendously computationally expensive to break – but not if the secret key is publicized!) In other words, anybody who wanted could fake the votes of any Accuvote TS machine anywhere.

Diebold then claimed this software was only experimental and was not the version used on produced and certified machines. That claim was a lie.

A December 2003 audit showed that Diebold illegally employed uncertified software on all the voting machines they sold to California counties. Even changing merely a few lines of code could of course cause completely different behavior of a computerized voting machine, so such code changes are illegal unless certified.[90] This audit makes it clear that law is almost entirely disregarded.

Voting activist Bev Harris [84] also discovered that Accuvote vote files could trivially be altered to any total values you like, by somebody without computer programming skills, using a program called "Microsoft Access" which activated automatically by clicking on the file's icon. The machine provides no paper audit trail and hence such changes would not be detectable.

Harris continued investigating and reported on 26 August 2004:

> The Diebold GEMS central tabulator contains a stunning security hole. Manipulation technique found... 1000 of these systems are in place, and they count up to two million votes at a time. By entering a 2-digit code in a hidden location, a second set of votes is created. This set of votes can be changed, so that it no longer matches the correct votes. The voting system will then read the totals from the bogus vote set. It takes only seconds to change the votes... This program is not 'stupidity' or sloppiness. It was designed and tested over a series of a dozen version adjustments.
>
> This problem appears to demonstrate intent to manipulate elections, and was installed in the program under the watch of a programmer who is a convicted embezzler.
>
> According to election industry officials, the central tabulator is secure, because it is protected by passwords and audit logs. But it turns out that the GEMS passwords can easily be bypassed, and the audit logs can be altered and erased. Worse,

[89] Actually, DES has not been secure for years. Typical delay to crack DES nowadays is under 22 hours, as indicated by `www.rsasecurity.com/rsalabs`'s "DES challenge III" which was cracked in 22 hours by the Electronic Frontier Foundation to earn a $10,000 prize in 1999.

[90] However, frankly, I dispute the entire idea of code "certification." In fact I do not believe that large code can be certified at a cost smaller than the cost of writing that software in the first place. Given that that is the case, the whole idea that a government should buy machines from a manufacturer and then certify their code (which they both keep secret), is ludicrous. It is especially ludicrous to imagine that individual US counties, acting in isolation, have what it takes to "certify" code. The only course that makes sense is for the government to develop the code itself, and make it public for certification by everyone.

> the votes can be changed without anyone knowing, including the officials who run the election.
>
> The MS Access database is not passworded and can be accessed illicitly through the back door simply by double-clicking the vote file. After we published this report, we observed unpassworded access on the very latest, GEMS 1.18.19 system in a county elections office.
>
> Some locations removed the Microsoft Access software from their GEMS computer, leaving the back door intact but, essentially, removing the ability to easily view and edit the file.
>
> However, you can easily edit the election, with or without Microsoft Access installed on the GEMS computer. As computer security expert Hugh Thompson demonstrated at the Aug. 18 California Secretary of State meeting, you simply open any text editor, like "Notepad," and type a six-line Visual Basic Script, and you own the election.

Florida 2004: In a special election held in early January 2004 in Broward County, Florida, with margin of victory 12 votes, the electronic voting machines used in that election failed to register any vote for 134 voters, even though there was only a single item on the ballot. In such a single-issue election, while it is certainly possible that a voter could go to the polling place, sign the log book, go to the touch screen machine, and choose not to cast any vote, it is somewhat hard to imagine that 134 voters would do so. "It's incomprehensible that 134 people went to the polls and didn't cast votes," said Broward County Mayor Ilene Lieberman. Since the voting machines in Broward County did not produce a voter-verified paper ballot, there is no way for elections officials to determine what really happened. Those 134 votes, if cast, are irrevocably lost. Such paperless electronic voting machines fail to comply with Florida Election Law which requires a manual recount of the ballots in small-margin elections, but that law is evidently disregarded.

Union county, Florida Sept. 2002: Machines read 2642 Democratic and Republican votes as 100% Republican. ES&S then paid for a hand recount, which fortunately was possible since Union County retained paper copies of ballots.

Allamakee County, Iowa 2000: (Reported in the *Wall Street Journal*.) An optical-scan machine was fed 300 ballots and reported 4 million votes for G.W.Bush (this exceeds Iowa's population all by itself). Retrying yielded the same result.

Comal County, Texas 2002: An election tabulated by ES&S machines gave 3 winning Republican candidates in a row each exactly 18,181 votes. This coincidence was not seen as reason to audit the election.

Conroe, Texas 2002: Congressional candidate Van Brookshire ran unopposed in the district 2 primary. He got zero votes. (Hadn't he voted for himself?) The ES&S computer had given all of his votes to U.S. Rep. Kevin Brady, who was unopposed for the nomination for another term in District *8*.

Wayne County, North Carolina 2002: Computerized machines skipped 5500 party-line votes, both Republican and Democratic. Fixing the error reversed the election for state representative from House district 11.

New Orleans, Louisiana 1994: Susan Barnecker lost an election, then demonstrated on a widely circulated videotape that on one touchscreen machine, votes for her were electronically recorded for her opponent. (This test was repeated several times.) Her protests were unavailing.

Fairfax, Virginia, November 2003: Testing ordered by a judge revealed that several voting machines subtracted one in every hundred votes for the candidate who lost her seat on the School Board. (This kind of error is very insidious.)

Maryland 2004: Michael A. Wertheimer, a consultant hired by the Maryland state legislature, found in January 2004 that "it is possible to vote multiple times, break into machines and disrupt results or get voters to select the wrong candidates. It's also possible to dial in to election headquarters and alter results or wipe out all of them." His team of hackers conducted an exercise Jan. 19 to simulate an attack on Maryland's Diebold touch-screen voting machines.

They found that individual machines could be disabled by jamming a voter card into a terminal or lifting it up and pulling out wires. The team guessed passwords on the cards that were needed to access the machines, and also found the passwords were contained in the source code of the computers. The computer server that tabulates election results did not have security updates from Microsoft Corp. Hence team members were able to break into the server remotely via dial-up modem.

He found each of Maryland's machines had two identical locks, which could be opened by *any* one of 32,000 keys. But this was not necessary since team members picked the lock in "approximately 12 seconds." They suggested that each voting machine have a different password. But Linda Lamone, administrator of the State Board of Elections, said that would be too risky and also said it was too late to equip all 16,000 Diebold machines with printers to provide paper copies of ballots.

Bottom line: Wertheimer's team broke into the computer at the State Board of Elections, changed the outcome of the (practice) election, left, and erased their electronic trail all in minutes.

Bob Urosevich, president of Diebold Election Systems, summed up Wertheimer's report as confirming "the accuracy and security of Maryland's voting procedures and our voting systems as they exist today" [156].

Dallas Texas 2002: 18 machines were pulled out of action in Dallas because they registered Republican when voters pushed Democrat. Republican judge Karen Johnson quashed audit attempts.

Florida 2000: In Volusia County, during the 2000 presidential election, the Socialist Workers Party candidate received almost 10,000 votes, about half the number he received *nationwide*; 4,000 erroneous votes appeared for G.W. Bush while at the same time, Presidential candidate Al Gore received *negative* 16,022 votes. This shot Bush, who appeared at the time to be losing Florida, to the front and was the direct cause of several television networks calling the election for *Bush*, which then caused Gore to concede nationwide defeat (a concession he later retracted).[91] But obviously, negative vote counts are not possible. This error was only spotted by alert Democratic poll-watcher Deborah Tannenbaum, who noticed Gore's vote total actually *de*crease by 16,000 votes – all due to precinct 216, which contained only 600 voters! This happened at 2am due to an upload of a second memory card (the results from the earlier upload of a first memory card were automatically and silently overwritten). So quite probably there were many other such errors which were not detected. (Possibly relevant: "We have a sordid history of election fraud in this [Volusia] county," Circuit Judge John Doyle wrote in a 1997 ruling [113].)

Observe that the Diebold tabulator had no problem with accepting an input of a negative number of votes, nor did it flag the discrepancy between the two memory cards.

Diebold Corp. emails discussing the problem surfaced and were posted by Swarthmore college students [158].[92]

```
From:  Lana Hires [mailto:lhires@co.volusia.fl.us]
Sent:  Wednesday, January 17, 2001 8:07 AM
Subject:  2000 November Election
```

```
...I need some answers!  Our department is being
audited by the County.  I have been waiting for
someone to give me an explanation as to why Precinct
216 gave Al Gore a minus 16022 when it was uploaded.
please explain this so that I have the information
to give the auditor instead of standing here "looking
dumb".  ...Any explantations you all can give me
will be greatly appreciated.  Thanks bunches,
Lana
```

What caused the problem? Here was the final emailed reply from from "Tab" (Talbot) Iredale, Vice President of Research & Development at Global/Diebold:

```
...The error could only occur in one of four ways:

1.Corrupt memory card.  This is the most likely
explanation for the problem but since I know nothing
about the 'second' memory card I have no ability
to confirm the probability of this.

2.Invalid read from good memory card.  This is
unlikely since the candidates' results for the
race are not all read at the same time and the
corruption was limited to a single race.  There is
a possibty that a section of the memory card was
bad but since I do not know anything more about
the 'second' memory card I cannot validate this.

3.Corruption of memory, whether on the host or
Accu-Vote.  Again this is unlikely due to the
localization of the problem to a single race.

4.Invalid memory card (i.e., one that should not
have been uploaded).  There is always the possiblity
that the 'second memory card' or 'second upload'
came from an unauthorised source.
```

The official explanation released to the press [113] was "damaged memory card" and the error supposedly was eventually corrected by re-uploading the first card and so all was well.

But, according to Diebold, an error due to a corrupt memory card should have been prevented by automated techniques involving 16 parity check bits, designed to make the probability that a damaged memory card could be used successfully, be 1/65536 (assuming it was still operating; otherwise the probability would be even lower). However, if the card's contents had been *intentionally* written onto the card, rather than caused by damage, then the probability of passing the checks would have been 100%. We also remark that, amazingly enough, Volusia's *total* number of votes was *exactly preserved* by the error because the negative Gore count was precisely compensated by the huge positive number of votes for independent/minor-party candidates (the largest such count in Volusia's history). For these reasons, I do not believe the official explanation.

I believe there are only two plausible explanations:

1. intentional fraud and
2. a machine crash caused writing to random memory locations. But, in this latter case it seems implausible that the total number of votes would be exactly preserved.

So the only remaining possibility seems to be (1). Notice that the Diebold manager (final line of his email) also agreed with me that it was a reasonable possibility that the second card was part of deliberate election-rigging conspiracy.

Nevada 2005: Nevada became the first state with e-voting machines with a voter-verified paper trail. Dean Heller, Nevada's Secretary of State: "otherwise it's a trust-me scenario, and I don't think that works today."

Hart-Intercivic and ES&S Inc. 2004: Hart-Intercivic and ES&S are also among the USA's largest voting machine manufacturers. The former was recently accused by one of its technicians of faking numerous test results and lying to numerous county elections officials.

Chuck Hagel, Republican senator of Nebraska, was the head of the company that owns ES&S, which installed, programmed, and largely ran the voting machines that were used by most of the citizens of Nebraska. (As of 2004, Hagel still has part ownership.) When Hagel ran there for the U.S. Senate in 1996, the *Washington Post* (13 Jan. 1997) said Hagel's "Senate victory against an incumbent Democratic governor was the major Republican upset in the November election." (Hagel's GOP primary victory was also an upset.) Hagel won virtually every demographic group, including many largely Black communities that had never before voted Republican, becoming the first Republican in 24 years to win a Senate seat in Nebraska. 80% of those votes came from his company's machines.

On 2 Feb. 2002, the *Baton Rouge Advocate* reported, "Arkansas Secretary of State Bill McCuen pleaded guilty to felony charges that he took bribes, evaded taxes and accepted

[91] In its internal investigation, CBS's inquiry team found the two Diebold County-level errors, Volusia and Brevard, were conclusive in their network's decision to call the race to Bush: "The mistakes, both of which originated with the counties, were critical, since there were only about 3% of the state's precincts outstanding at this time. They incorrectly increased Bush's lead in the tabulated vote from about 27,000 to more than 51,000. Had it not been for these errors, the CBS News call for Bush at 2:17:52 AM would not have been made."

[92] Swarthmore and the students were then threatened by Diebold lawyers who claimed they were violating "copyright." The Swarthmore students sued and won – Diebold was found by the judge to have knowingly falsely claimed copyright protection. It is now liable for up to $5 million in penalties.

kickbacks. Part of the case involved Business Records Corp. [now merged into ES&S]... Arkansas officials said the scheme involved ... then-BRC employee Tom Eschberger .... Eschberger got immunity from prosecution for his cooperation." Eschberger later became vice president of ES&S.

Sequoia voting machines. In 1999, two Sequoia executives, Phil Foster and Pasquale Ricci, were indicted for paying Louisiana Commissioner of Elections Jerry Fowler an $8 million bribe to buy their voting machines. Fowler is currently serving five years in prison.

"Secret" Sequoia voting machine code was found on an open web site [84].

Election.com: Was a US voting machines company; a controlling interest in it was owned by Saudi Arabians.

Washington DC, August 2004: "A four-day conference for election officials was held, co-sponsored by voting machine vendors who want their business...

The plans for the event included women in evening dresses and men in tuxedos carrying a six foot-long check made out to 'election officials' for 'parties, cruises, wining and dining' and signed by voting machine manufacturers Diebold Elections System, Sequoia Voting Systems, and Elections Systems and Software. " [126]

9.7 Vote counting

Iraq 2002: Iraq headman Saddam Hussein claimed to have won a 100 percent "Yes" vote in an October 2002 referendum on a new term.

San Fransciso, California 2001: Bright red ballot box lids were found floating in the bay and washing up on ocean beaches for several months after the November 2001 election.

New Mexico 2000: GOP activist Shelley Hayner and 11 volunteers audited the votes in Dona Ana County, New Mexico [68]. They found that 5509 absentee ballots had been submitted with signatures, yet 6456 were counted in the November 2000 final, official tally. When journalist Federico Almarez asked Denise Lamb in the secretary of State's office to explain the 947 phantom votes, she blamed "administrative lapses."

The *Washington Post* examined Rio Arriba county [98] and found that 678 of the county's 2300 voters did "not" cast a vote for any Presidential candidate. In one district, official totals showed 203 votes, but 0 votes recorded for either Gore or Bush. These problems were caused by electronic voting machines.

Al Gore "won" New Mexico by 366 votes.

North Carolina 2004 [120]: The state was unable to swear in an agriculture commissioner because Steve Troxler and Britt Cobb were 2287 votes apart while an electronic voting machine in Carteret County lost 4438 votes. "The machine had mistakenly been set to keep roughly 3000 votes in memory, which was not enough, and in a spectacularly bad design decision it was programmed to let people keep 'voting' even when their votes were not being saved." This "prompted North Carolina to reconsider its use of paperless electronic voting." The 3000-vote memory limit had been incorrectly claimed by the voting machine manufacturer to be 10000. After this election, a re-election was held in Carteret County just for agriculture commissioner, but that re-election was thrown out in a court decision. Then a statewide agriculture commissioner re-election was held but also thrown out in a court decision. Finally the matter was resolved by collecting 1352 affadavits from voters who claimed to have voted on that machine.

USA, 2000: US House member John Conyers compiled reports from 31 states and the District of Columbia and found that 1,276,916 voters had had their votes discarded (i.e. unused in the 2000 Bush-Gore presidential contest). This exceeded the nationwide difference in the popular vote between Gore and Bush. Conyers recommended, at a minimum, that voting machines notify voters *immediately* if their ballot is invalid (due to undervotes, overvotes, or stray pencil marks). As of 2004, this recommendation remains merely that.

9.8 Three fraud-aided US Presidents

George W. Bush, 2000: Bush won the presidency over Al Gore despite winning 543,895 fewer popular votes. This was thanks to the electoral college and in particular thanks to the decisive state of Florida which he won by an official margin of 537 votes over Gore.

We have several comments to make on this. First of all, it was completely impossible for Florida to count votes accurate to ± 537. The *Miami Herald* found thousands of known illegal votes (by felons, dead people, out of state residents, etc.) and (thanks to ballot secrecy) it is unknown for whom they were cast. Therefore, if the "true" margin was 537, then it was mathematically impossible to determine the winner.

Second, it is now clear that an illegal tactic was employed which artificially swung the vote in Bush's favor by at least 20,000. Bush was aided by the facts that (1) Jeb Bush, his brother, was governor of Florida. (2) Katherine Harris, his Florida campaign manager, was the Florida Secretary of State, i.e. the top elections-supervising official in the state.

Before the 2000 election, Harris ordered county elections officials to purge 57,000 citizens from voter registries as "felons" not allowed to vote in Florida. But actually about 95% of them were innocent of crimes – but 54% were guilty of being black. (Statewide, blacks supported Gore by a 9:1 ratio, according to *The Washington Post* 31 May 2001.) DBT On-Line, the company which had been paid over 400 times the previous company's rate to prepare the list (and was awarded the contract with no competitive bidding), after being sued by the NAACP, turned over to the NAACP's lawyers a report indicating that the state ordered a purge of 94,000 "felons" specifically requesting *not* performing rudimentary checks such as social security number matching (although checks on the voters' race *were* to be employed) despite the fact that according to the company's data, no more than 3000 were likely illegal voters [122].

If only the *actual* felons on that list had been excluded from voting, then there would have been about 32,000 additional Gore votes and about 12,000 additional Bush votes. (Contrast this with Bush's official 537-vote margin.)

Third, tactics which perhaps were not illegal, but certainly were reprehensible, gave Bush an additional artificial 70,000-vote advantage.

In the 4 blackest counties in Florida, 7-12% of all ballots were rejected as invalid (due to stray marks, overvoting, or undervoting) and not incorporated into the count in the Bush-Gore contest. In the 4 whitest counties, the rejection rate was 0.5-2%, i.e. 5 times smaller [122].

This was largely due to the fact that the voting machines in the blackest counties were set to reject invalid ballots *silently*, with no indication to the voter that anything was wrong; the opposite was true in the whitest counties. (Sample voting machines from the counties were on display outside governor Jeb Bush's office for some time before the election so that the higher-ups could be sure they were adjusted right.)

This disparity occurred not only on a county level, but also on a neighborhood level [14]: ballots cast from black neighborhoods throughout Florida were four times as likely to go uncounted as those from white neighborhoods. Nowhere was the disparity more apparent than in Duval County, where 42% of the 27,000 ballots thrown out came from four heavily Democratic black precincts. (That *alone* represented a $\approx$ 5000-vote boost for Bush over Gore. Keep remembering: Bush won Florida over Gore by 537 votes.)

Countervailing pro-Gore fraud? We have explained how Bush gained 20,000 votes versus Gore in Florida due to the fraudulent Felon's list and 70,000 due to systematic suppression of black votes. However, some Republicans believe there was also pro-Gore fraud. The two main hypotheses that have been advanced are that (a) about 1443 extra votes for Gore over Bush were "found" during recounts by Democrat-biased county election committees, (b) Massive fraud by punching of Gore holes through entire reams of "butterfly ballots" in Palm Beach County (which would have no effect on Gore votes, but would invalidate all non-Gore votes as 'double' votes) generated 50,000 extra Gore-over-Bush margin. (Observe that this fraud technique would not work with most other kinds of ballots.)

To this we reply: (a) If so this was evidently not enough to tip the election and was picayune in comparison. (b) This is a more serious charge. It was raised by Robert A. Cook (a "Nuclear Engineer with an M.S. in statistical quality control") in numerous internet postings. Palm Beach County had the second highest percentage of invalid ballots (a 4.2% overvote rate) behind Duval County, and, Cook claimed, it was the only Florida county with fewer Bush votes (152,954) than registered Republican voters (231,626). Suspicious. *But*, Cook's charges fall to the ground when one considers the following facts [7].

1. The total number of overvotes in Palm Beach County was 19,235.
2. The *Palm Beach Post* inspected them all and found Gore appeared on 15,371 (80%), Buchanan on 8689, McReynolds on 4567, Browne on 4218, and Bush on 3751.

This data is totally incompatible with Cook's theory and evidently could have cost Bush *at most* 3751 votes. In fact, the obvious explanation of Palm Beach's high overvote rate was its notoriously poorly designed "butterfly ballot" which caused a large number of elderly voters to mistakenly vote for Buchanan instead of Gore. Palm Beach County generated 3411 Buchanan votes, exceeding the next highest Florida county by a factor of more than 3. Meanwhile absentee voters (who used a different ballot) in Palm Beach voted Buchanan at a 4-times smaller rate, although the Buchanan proportion did not differ appreciably between election-day and absentee ballots in any Florida county besides Palm Beach. This makes it clear that about 2500 of Buchanan's votes actually were intended for Gore. The butterfly ballot also engendered misvotes for McReynolds. And indeed McReynolds received 302 Palm Beach votes, about 10 times what would be expected based on his statewide totals. So really, the truth is probably that *Gore* deserved an extra 13,000 margin over Bush from Palm Beach.[93]

The *Miami Herald* and other newspapers analysed 111,000 overvotes Florida-wide and found that Gore was marked on 84,197 and Bush on 37,731, which strongly suggests that any overvoting fraud hurt Gore far more than Bush. In short: Cook is completely wrong; any pro-Gore fraud in Florida was tiny in comparison to pro-Bush fraud.

Postscript: K.Harris and Florida in 2002 admitted that tens of thousands of black voters had been wronged, and agreed to return them to the voter rolls *at the beginning of 2003*, i.e. *after* counting the ballots in Jeb Bush's re-election race.

In 2004, the news organization CNN requested to examine Florida's new felons list before the 2004 election, but that request was denied. After they won that permission on 1 July 2004 in a lawsuit, the list – again with every one of the over 47,000 voters on it identified by race – was found, amazingly, to contain fewer than 0.1% Hispanics, in a state where nearly 1 in 5 residents is Hispanic. (Florida Hispanics predominantly vote Republican. Remember, over 50% on the 2000 list had been black, in a state with 12% blacks. Black voted 9:1 for Gore.) The *Miami Herald* then reported on July 2 that it found more than 2100 names erroneously included on the list because they had received clemency. According to Jeb Bush, these things were "an oversight and mistake." Nine days after the list's release, state officials decided to scrap it entirely, saying it was too flawed to be trusted. However, Florida intends to use the list again in 2006.

Lyndon Johnson and J.F.Kennedy, USA 1948-1960: Then-congressman Lyndon B. Johnson, later to become US president, won his Senate seat from Texas in 1948 by 87 votes over former Gov. Coke R. Stevenson, the closest senatorial election in history.

Stevenson had bested Johnson in the Democratic primary, but received only a plurality, so a run-off was required. Since there was no chance that a Republican could be elected statewide at the time, the winner of the primary would be assured the Senate. An unofficial tabulation showed Stevenson in the lead by 114 votes out of nearly 1 million. But then, suddenly, 6 days late Precinct 13 of Jim Wells County, part of an 11-county region in South Texas controlled by pro-Johnson political boss George B. Parr (1901-1975), mysteriously "found" a ballot box containing an additional 203 votes – 201 for Johnson and 2 for Stevenson – giving Johnson his 87-vote statewide lead and the victory 494191-to-494104. J. Evetts Haley [81] noted that the 202 who voted for LBJ "had been added [to the recount

[93] Also, if entire reams of ballots were punched through – then why did nobody find an example ream that *had* been entirely punched through?

list] alphabetically in blue ink, whereas the original list was in black." All of the 203 names were in "the same handwriting." Other voters didn't live in the county anymore or were dead. (The list eventually was mysteriously "lost.")

The Precinct 13 election judge, Luis Salas, had absolute say over the vote counts in the Hispanic South Texas precinct. In 1977, Salas, then 76, sought "peace of mind" by admitting that he had certified enough fictitious ballots to steal the election. The Toledo (Ohio) Blade for 31 July 1977 quoted Salas as saying: "Johnson did not win that election; it was stolen for him. And I know exactly how it was done."

99.6% of the eligible voters in Texas's Duval County (Parr's headquarters) voted, and they voted for Johnson by a 100:1 ratio.

All these things were not a coincidence. The story of how Johnson, with heavy financing from the Brown and Root Tobacco company, paid off Parr to create votes for him, is recounted in detail in Johnson's biography [33]. (Another good account of all this is in [60].) Parr and his wholy-owned election-judges and sheriffs would march Mexicans into the polls, pay their poll taxes for them, hand them their pre-filled-in ballots, and tell them to drop them in the box, afterwards rewarding them with a drink of tequila. Too-inquisitive election inspectors and observers would be jailed and ordered out of the county at submachine gun point; the homes and businesses of those insufficiently loyal to Parr would be burned. Three prominent critics of Parr were assassinated by unknown assailants. On the 1948 election, Parr's counties simply kept "finding" more votes each day after the election was over until, a week afterward, Johnson finally was in the "lead."

In 1954, Johnson re-won his Senate seat, this time by a comfortable 500,000 votes.

In 1960, J.F.Kennedy (with LBJ his running mate) won the presidency from R.M.Nixon by 118574 votes out of more than 68.8 million cast [32]. JFK had 303 electoral votes to Nixon's 219 (269 needed to win). Had Nixon carried both Illinois (which JFK carried by 8858 out of more than 4.7 million votes cast) and Texas (46,242 votes out of over 2.3 million cast), he would have won with 270 electoral votes[94].

There is strong reason to suspect that both the Texas and the Illinois victory – and perhaps Kennedy's national popular vote plurality too – were due to fraud. Chicago was controlled by its notorious political boss – the longest serving mayor in US history – Richard J. Daley. (Daley was also the father of Gore's 2000 campaign manager William Daley.) Texas too was controlled by Democrats.

The turnout in Chicago was a spectacular 89%. This contrasts with the nationwide turnout of 63%. It also contrasts with the fact [62] that in the 11 presidential elections during 1960-2000, totalling 550 statewide contests, *not once* did any state ever exceed 78.4% turnout (Utah 1964), and the states with the largest-% turnout were always rural (namely North and South Dakota, Utah, Minnesota, and Maine), not urban.

Despite losing 93 of the 102 counties in Illinois, Kennedy won the state by 8858 votes thanks to his 456,312-vote advantage in Chicago, whose precincts reported their totals remarkably late. (Compare this with Kennedy's *nationwide* plurality of 118,574.)

Mayor Daley defended Chicago by claiming Democratic fraud there was no worse than Republican fraud in downstate Illinois [36]: "You look at some of those downstate counties," he said, "and it's just as fantastic as some of those precincts they're pointing at in Chicago."

In 1962, after an election judge confessed to witnessing vote tampering in Chicago's 28th ward, three precinct workers pled guilty and served short jail terms[95]; 677 others were indicted before being acquitted by Judge John M. Karns, a Daley crony. Many of the allegations involved practices uncorrectable by a recount, leading the *Chicago Tribune* to conclude that "once an election has been stolen in Cook County, it stays stolen."

Earl Mazo was the Washington-based national political correspondent for the *New York Herald Tribune* and wrote a series of articles on election frauds. Mazo went to Chicago, got lists of voters in suspicious precincts, and started checking their addresses. "There was a cemetery where the names on the tombstones were registered and voted," Mazo recalls. "I remember a house. It was completely gutted. There was nobody there. But there were 56 votes for Kennedy in that house." In Ward 27, Precinct 27, 397 votes were recorded from 376 voters. Mazo then went to Republican areas downstate and looked for fraud there. (Practices there included voting by telephone and bulk voting by political leaders.)

"In downstate Illinois, there was definitely fraud," he says. "The Republicans were having a good time, too. But they didn't have the votes to counterbalance Chicago. There was no purity on either side, except that the Republicans didn't have Daley in their corner – or Lyndon Johnson."

The *Chicago Tribune* stated "the election of November 8 was characterized by such gross and palpable fraud as to justify the conclusion that [Nixon] was deprived of victory." (As quoted by the Washington Republican National Committee, who filed a lawsuit challenging the Chicago results.) The case

[94] We shall argue that Nixon probably would have won Illinois and perhaps Texas with honest voting and counting. However, this does not necessarily mean Nixon should have won the presidency, because if any of *Nixon's* state-victories were also due to fraud (or mistakes) – which is not implausible and which is an issue that has essentially not been studied – then correcting them would have given the presidency back to Kennedy. And indeed Hawaii, after a court decision ordering a recount, changed its victor from Nixon to Kennedy on 28 December. Judge Ronald B. Jamieson ruled that Kennedy had won by 115 votes out of about 185,000 cast; the original official tally had said Nixon won by 141. One of the discrepancies that turned up in the early stages of the recount and encouraged the judge to continue it was precinct 17 of district 15 of Manoa Valley, where 69 more votes had been tallied than the number of voters; 68 of these 69 phantom votes were for Nixon. There is no doubt that both Democratic and Republican vote fraud occurred; it is just that in Illinois and Texas the Democrats were far more successful at it.

[95] Seymour Hersh further claims in his book *The dark side of Camelot* that the Kennedys enlisted crime boss Sam Giancana to acquire more votes as part of a deal in which it was agreed to "cut the heat" on him from law enforcement. Giancana's own view, expressed to Judith Campbell Exner, the mistress he shared with JFK, was "Listen baby, if it wasn't for me your boyfriend wouldn't even be in the White House" ([133] p.214). in 1992 Giancana's nephew and brother wrote a book [74] again recounting how Giancana had rigged the Cook county vote for Kennedy as part of a deal, and further stating that when Kennedy reneged on the deal, Giancana had him assassinated. (They claimed they had heard this directly from Giancana himself who also noted "Richard Nixon and Lyndon Johnson knew about the whole damn thing." The book also linked Giancana to a total of 7 US Presidents.)

was assigned to Circuit Court Judge Thomas Kluczynski, a Daley machine loyalist. On Dec. 13, Kluczynski dismissed the Republican suit. Less than a year later, on Mayor Daley's recommendation, Kennedy appointed Kluczynski to the federal bench.

After Kennedy took office, he appointed his brother Robert F. Kennedy Attorney General, i.e. head of the Justice Department and thus indirectly of the FBI. This appointment was quite odd considering RFK's limited lawyerly experience. (He never tried a case in a courtroom in his life.) The Justice Department then advised the FBI to cease investigating election fraud charges.

Meanwhile, Fannin County in Texas had only 4895 registered voters, but 6138 votes were cast (125% turnout!), 75% for Kennedy. In Angelina County, in one precinct, only 86 people voted yet the final tally was Kennedy 147, Nixon 24. And so on. The population of the 11 Parr-controlled counties was about 280,000, which alone would have been enough to explain about 34,000 votes worth of Kennedy-Nixon margin (based on the 2.3 million votes in Texas versus its 9.6 million population, and assuming a 50:50 vote split was converted by Parr into 99:1; compare this 34,000 with the official 46,242-vote margin)[96] The Republicans demanded a recount, claiming that it would give them 100,000 votes and victory. John Connally, the state Democratic chairman, said the Republicans were just "haggling for headlines" and predicted that a recount would give Kennedy another 50,000 votes. (Observe that both of these estimates exceeded the official margin.) But there was no recount. The Texas Election Board, composed entirely of Democrats, had already certified Kennedy as the winner.

9.9 Election fraud as a government-toppling or democracy-destroying event

Indira Gandhi, India 1975: After a court ordered Prime Minister Indira Gandhi removed from office because she was elected with the aid of election fraud, Gandhi responded by refusing to step down, jailing thousands of both her political rivals and journalists she disliked, dissolving many state governments, and terminating Indian democracy in favor of rule by decree. Fortunately the resulting period of authoritarian rule only lasted for 19 months. But the larger point is: *election fraud, or even just perceptions of it, has the power to put an end to democracy.*

Costa Rica 1948; Nigeria 1965: Costa Rica's Civil War (1948), which led to a new government in 1949 (with a new constitution, and awarding the vote to Blacks and women) started with allegations of election fraud and with the sitting government refusing to accept the election result. Although both the Indian and Costa Rican examples led to an ultimately happy conclusion, that was not the case in Nigeria where this same scenario led to a civil war in 1965 followed by military rule interspersed with brief periods of pseudo-democracy for the next 34 years.

9.10 Conclusions

Election frauds and/or allegations thereof have toppled democracies. In the US, three presidents during 1950-2000 arguably got there with the aid of vote fraud comparable to or greatly exceeding winning margins.

Gore's concession (later retracted) in the 2000 race was caused by CBS News' assessment that Bush won Florida, which in turn was directly caused by an electronic voting machine "error" which (a) was outrageous – a negative "vote count" was accepted automatically without any flag being raised – and (b) whose most likely explanation was intentional fraud.

Australia has adopted nationwide standards for electronic voting machines and their computer code is *open-source*.[97] The USA in contrast has many competing voting machine vendors whose code is a legally protected trade *secret*, and usually, in practice, changes illegally during and before elections without any certification. Many electronic voting machines permit magic alteration of vote totals with no trace.

10 Will quantum computers destroy cryptographic election protocols?

If quantum computers ever are built, then it will become possible to factor N-digit integers and solve N-bit elliptic curve (or mod P) discrete logarithmic problems in polynomial(N) time [143]. That in turn would destroy all the usual public-key cryptosystems and would crack most of the cryptographic tools and protocols we discuss in this survey.

We have several reassurances to those who fear this horrfying possibility:

1. I doubt that quantum computers ever will be built, and regard all claims of partial success in that duirection as consisting mainly of hype.
2. I feel certain that progress toward build a quantum computer will be very slow, and so we would be forewarned many years in advance that big progress is finally beginning to become a worry.
3. It will still be possible (albeit more painfully) to perform cryptographic voting protocols even when quantum cryptography comes – and we will now explain how.

First, our general purpose theorem 3 about ZK-proofs of NP-statements is based entirely on bitstring "commitments," which may be implemented entirely by means of AES-type cryptography, without need of either public-key cryptography or any assumptions that discrete logarithms are hard. *These* cryptosystems are (as far as anybody knows) immune to quantum computers, except possibly for a need to double the keylength to avoid succumbing to Grover's quantum-speedup of brute-force search [165].

Therefore, all zero knowledge proofs can still be implemented (perhaps more painfully, but it can be done) to be secure against enemies with quantum computers.

Second, all known schemes for secure multiparty computation (SGMPC, see §4.27) unfortunately depend on the assumption

[96] Parr, facing jail on federal tax-evasion charges, committed suicide in 1975, whereupon his political machine's power finally began to wane.

[97] We hasten to remark, however, that open-source code is not a panacea, since Ken Thompson [160] has demonstrated language-redefinition techniques that allow software to do anything *despite* the source code looking completely innocent.

that discrete logarithms are hard. However, it is known [?][?] that Shamir secret sharing and hence SGMPC can be implemented in such as way as to be secure *unconditionally*, i.e. information-theoretically without need of *any* unproved computational complexity assumption, *provided* that the distrustful computing parties are allowed to communicate, not only via a broadcast channel, but also via untappable channels between any pair of the parties. And it is possible to create an untappable communication channel by use of "quantum channels" based on transmission of bits via single photons – this has also been called "quantum cryptography" [16][17] – Indeed, such channels both have been built[98] for communication over optical fibers over distances of ≤ 100km, and found to perform quite well. Free-space quantum bit transmission has also been demonstrated, e.g. via telescope and laser between the Max Planck Institute and the Zugspitze 21km away [157].[99]

Essentially everything else we have discussed is just a special case of one or both of these two general purpose theorems (albeit often a much *faster* special case). We conclude that, aside from a possible slowdown by a possibly-large constant factor, everything in this paper could be transmuted into some form which will survive even after the hypothetical day when quantum computers become a threat.

Third, any public-key cryptography used for, e.g. key-exchange, could be dispensed with if the parties involved could simply communicate via untappable quantum communication channels.

The task of designing *good*, i.e. work-efficient, quantum-computer-immune protocols (as opposed to merely proving that ones *exist* without striving for efficiency) will be left to future authors!

11 Conclusions

11.1 Ten lessons

1. Good methods exist. We have demonstrated that election methods can be constructed which simultaneously obey *all* of the desiderata of §2, despite the naive impression, even among experts such as Rebecca Mercuri, that many of these desiderata conflict.

2. Forget mixnets. Although both are good enough to reach engineering feasibility, secure voting schemes based on *homomorphic encryption* are inherently superior to those based on mixnets. The latter idea should be dropped.

3. Use elliptic curve cryptosystems. Elliptic curve based public key systems are far superior to non-elliptic ones.

4. Forget Chaum's "secret ballot receipts" scheme. Chaum's "secret ballot receipts" voting scheme [35] (criticized in footnote 60) *permits* vote selling and does *not* yield ballot secrecy (despite his claims) and hence should be discarded.

5. Voters must have inseparable personal "digital assistants." It is inherently impossible to achieve the election desiderata of §2 if voters are forced to vote on computerized voting machines provided by the election authority. (This was, in fact, the problem with Chaum's [35] scheme.) That is because the voter would (1) have to provide his identity so that his name could be "crossed off the list" of eligible voters who have not yet voted[100] (2) have to enter his vote in some human-readable (as opposed to encrypted) form, and the voting machine then would be free to *remember* that vote, thus violating the secret ballot principle. For this reason (and since the necessary cryptographic calculations are far too great for voters to perform manually, and since 300-digit crypto keys are too much for voters to be expected to memorize) it will always be necessary for the voter to interact with the voting machine solely through the intermediary of his *own* "digital assistant" (personal computer or "smart card") and it will always be necessary for that assistant to be physically protected (our mathematical treatment regards the assistant as a "part of the voter").

A few remarks on smart cards. The fact that every voter must have such a smart card is a major hurdle faced by secure voting schemes. Would such cards be cheap enough, reliable enough, and not too-heavily "spoofed"? Would there be considerable card-theft motivated by the desire to deny votes? Voting machines in districts containing a lot of opposition-voters could "accidentally" fry people's smart cards by a high voltage as soon as they were plugged into the machine. To prevent that the cards instead would have to interface *optically.* The card's optics could also double as a barcode scanner, useful for reading the voter's printed receipt and verifying its validity immediately. Each voter could download standard ballots and then preprogram his card with his vote *before* entering the voting booth, speeding matters up.

6. Privacy is required? It seems inherently impossible to achieve the election desiderata of §2 if voters can vote over the internet from locations of their choice. That is because voters would then be free to have others witness them in the act of voting, and hence could sell their votes. If so: to prevent that, it will always be necessary for each voter, at some stage, to communicate with the election authority in a secure and private location, e.g. a voting booth.

However: there are two methods which can evade this impossibility argument to a considerable extent. First, we can set up ways for voters to cast fake votes, so that the vote-buyer or vote-coercers could not know if the voter they were witnessing was casting a genuine or fake vote. However, the voter would have to prepare to make that choice, and the vote-buyer could witness the voter during that preparation.

[98] In May 2005 in Cambridge England, Toshiba gave a public demonstration of an untappable quantum communication channel capable of transmitting bits for distances of over 100km over an optical fiber, and indeed demonstrated secure video and voice transmission over that channel. Dr. Andrew Shields led the Toshiba group developing the system.

[99] Incredibly, as R.J.Hughes showed in pioneering experiments near Los Alamos, by transmitting the single photon in a very narrow time window and over a narrow beam from a telescope, it is possible to receive it fairly reliably even in broad daylight (although fog and rain can eliminate transmission). Lost single photons, even if they all are received by an eavesdropper, do not matter; thanks to an overarching algorithmic protocol which also involves bits transmitted in both directions over a nonquantum broadcast channel [16], the security of the transmission is not affected. The Toshiba team introduced the technique of delineating that time window via a bright "guide" laser flash.

[100] To see that deduction of the voter's identity is necessarily possible, consider the last voter to vote, after all other eligible voters already have. Or consider the sole voter at some remote voting location.

So at *some* point – either the voting or the preparation – voter privacy is required. More simply, it is possible to allow voters to vote multiple times, with only the last vote counting. This arguably would prevent vote selling and coercion unless the buyer/coercer could be sure his bought/coerced vote was absolutely the last one that voter would issue. So it would suffice for the voter to get private access at *some* later occasion before voting ended. Either of these ideas is possible in the JCJ scheme [96] (see §7.4). These two ideas might not prevent vote buying and coercion, but certainly would reduce it. Nevertheless, because off-site voters could simply sell their entire computer system to a buyer, who could then use it to vote in their name, or use his possession of it to deny them the ability to vote (or at least to make it more difficult), vote buying or coercion would still be possible to a considerable extent.

7. Multiparty computation schemes: great in principle, but presently impractical. Although we have demonstrated that, in principle, secure general multiparty computation schemes can render any vote-combination method secure, just the brute force use of that plan requires too much computation and communication to achieve engineering feasibility with present technology. Therefore *no* feasible secure election methods are presently known for Hare/Droop-STV [161][110] and reweighted range voting [149] – two of the three best voting methods currently known for multiwinner elections. This perhaps is an argument in favor of asset voting [151], an unconventional, but nevertheless perhaps also good, multiwinner election method. (Asset voting is additive and hence can be handled by homomorphic encryption.)

8. Multiparty schemes are necessary with fully general vote-combination methods. It is inherently impossible to achieve the election desiderata of §2 if *fully general* vote combining methods are allowed and if a *single* election authority carries out all the computations. Secure general *multi*party computations using "shared-secrets" in principle can do it, but with present technology that seems infeasible.

9. Bogus registrations. The usual crypto-secure schemes do *not* protect against the State creating a ton of fake voters, registering them, and then having them "vote." That whole cheat process could be automated and is one of the larger weaknesses of the present scheme. It also does not protect against the state declaring a ton of voters "ineligible." Both have been popular forms of cheating throughout American history including recent years.

Nevertheless, the system we advocate would be far superior to present ones, since there would be a world-readable posted list of registered voters. it would therefore be easy for anyone to check the list to try to find fake ones (such as dead ones, ones not at the claimed address, etc) and if the percentage of such were large enough, then they could not escape detection. (If the percentage were too small, then they could not affect the election much.) Further, any large number of voters denied registration could easily prove that. In the scheme of §7.3 any voters proven to be bogus could easily be deleted along with their votes, and the election then rerun ex post facto – *vastly* superior correctability to present methods.

10. Biased election officials. In both the 2000 Bush-Gore and 2004 Bush-Kerry presidential contests, the supreme election official in the crucial state was Bush's state campaign chair! This should be illegal.[101]

11.2 What we can and cannot do

It is important to understand what secure election schemes do and what they do not do.

What they do. They start with a publically posted list A of eligible voters and a publically posted list B of (encrypted signed dated) votes gotten from voters. They combine these votes to produce the election result, which they then announce. If the election authority and associated entities follow the protocol, then it will also produce a *proof* that[102]

1. Only those voters who know the private keys corresponding to the voter public keys listed in A could have produced votes on B,
2. nobody successfully double-voted,
3. every vote on B would be (once decrypted) legitimately formatted, i.e. valid,
4. the election result was correctly calculated from B, and the scheme also has the properties that:
5. every voter can confirm that his vote appears on B,
6. everyone can determine which voter's votes appear on B,
7. nobody can determine anything about what (plaintext) vote any voter produced (except of course that the voter himself knows it, and except insofar as that information is deducible from the election result itself).

If, on the other hand, they do not follow the protocol, then no such proof will be produced and it will be publically apparent who first violated the protocol.

What they do not do. The above guarantees are essentially of the form "if the input is right, and the protocol is followed, then the output will be right, and this will be proven without revealing anything about the input that is supposed to be kept private. And if the protocol isn't followed, we'll all know it." While wonderful, these guarantees are not omnipotent.

They are *not* of the form "the input is right." That is a human rather than a mathematical problem.

Thus if the original list A of "eligible" voters was obtained by some unfair or illegal process (for example, refusing to let anyone with dark skin register, cf. §9) or if black voters were physically prevented from contributing their vote to B, or if some class of gullible voters (where in practice, by "'voter" we mean "the entity consisting of both the voter and his 'digital

[101] The *Columbus Free Press* reported that (according to their anonymous sources) US President G.W.Bush had met with Ohio election commissioner K.Blackwell on the 2004 election day. Not only that, Blackwell was openly simultaneously serving as the Bush-Cheney Ohio campaign co-chair. Contrast this with the situation in the Phillipines in 2005 [134]: President Gloria Arroyo admitted talking on the telephone to an election official during vote-counting of the close May 2004 election; she later said this was a "lapse in judgement." That caused a scandal, calls for her resignation including from her own cabinet, and the initiation of impeachment proceedings.

[102] Subject to assumptions about the infeasibly great computational difficulty of certain problems related to discrete logarithms, and assumptions that certain sets of entities will refuse to collude, and assumptions that the votes in B were input from each voter under private unrecorded unwitnessed circumstances

assistant' pocket computer") provided votes they did not intend to provide because they were fooled, then the procedure will still produce the "correct" election result, with proof, for the input that *was* received. (Eligible voters who did not vote could, however, prove that fact.)

Furthermore, they are *not* of the form "the protocol will be followed." The election authority could willfully refuse to follow the protocol, or might be prevented from doing so by some kind of attack. In that case, the best we can say is that we would *know* it – it would not be possible to not follow the protocol and pretend it did. We can and have constructed systems with a certain amount of robustness against attacks, in the sense that there is recountability from paper records, that the system can survive a temporary breakdown of communications, and that voters whose vote does not appear on B can try voting again with no penalty until it does appear. (Voters also could refuse to follow their protocols, in which case they would be unable to vote.)

Speaking purely as a computer programmer, though, the problem is solved, in the sense that we have a procedure for converting inputs (alleged votes) into outputs (election results) in a way verifiable by anybody and which satisfies the desiderata about voter privacy, nonmanipulability, etc. The problems we've mentioned are not the concern of the computer programmer – they are merely human problems about *obtaining* the inputs. The computer programmer's job begins once those inputs *have* been obtained.

Furthermore, it seems as though any election authority trying to be unfair in the ways we have mentioned, on any scale large enough to be useful, in any sufficiently open society to be having elections and using this sort of secure system in the first place, would necessarily be detected (cf. "lesson 9" in §11.1) and the error would then be correctable. For example, anybody trying to manufacture and distribute many fake smart cards, would run a big risk of detection (cf. "reply 1" in §8); so would any attempts to create or delete any large percentage of voters from the publically posted registration rolls.

References

[1] Masayuki Abe: Universally verifiable MIX net with verification work independent of the number of MIX centers; pp. 437-447 in EuroCrypt 98, Springer Verlag LNCS #1403.

[2] Masayuki Abe: Mix-networks on permutation networks, ASIACRYPT (1999) 258-273, Springer LNCS #1716. Masayuki Abe & Fumitaka Hoshino: Remarks on Mix-Network Based on Permutation Networks, Public Key Cryptography (2001) 317-324, Springer (LNCS #1992).

[3] M. Abe & E. Fujisaki: How to date blind signatures. In: Asiacrypt 96, LNCS 1163, pp. 244-251.

[4] Alessandro Acquisti: Receipt-Free Homomorphic Elections and Write-in Voter Verified Ballots, CMU-ISRI-04-116, May 2004, available at `http://www.heinz.cmu.edu/~acquisti/research.htm`.

[5] AES, Advanced encryption standard: `http://csrc.nist.gov/CryptoToolkit/aes/`.

[6] Manindra Agrawal, Neeraj Kayal, Nitin Saxena: PRIMES is in P, `http://www.cse.iitk.ac.in/news/primality.html`.

[7] Alan Agresti & Brett Presnell: Misvotes, undervotes and overvotes: The 2000 presidential election in Florida, Statist. Sci. 17,4 (2002) 436-440

[8] AP News: Fla. County Says Absentee Ballots Missing, 27 Oct. 2004.

[9] A.O.L. Atkin & R.G. Larson: On a primality test of Solovay and Strassen, SIAM J. Comput. 11,4 (1982) 789-791.

[10] E.Bach: Toward a theory of Pollard's rho method, Information & Computation 90 (1991) 139-155.

[11] E.Bach & J.Shallit: Algorithmic Number Theory, Vol. 1: Efficient Algorithms. Cambridge, MA: MIT Press, 1996.

[12] Eric Bach: How to generate factored random numbers, SIAM J. Computing 17,2 (1988) 179-193 (special issue on cryptography).

[13] Jo Becker & David Finkel: Now They're Registered, Now They're Not Election Officials Express Dismay at Extent of Misinformation, Variety of Tricks Targeting Voters, Washington Post, Sunday 31 October 2004, page A22.

[14] Jo Becker: Pushing to Be Counted in Fla. Groups Say That Blacks May Not Be Heard at Polls, *Washington Post*, Wednesday 13 October 2004, front page.

[15] M.Bellare & S.Micali: Non-interactive oblivious transfer, pp. 547-557 in CRYPTO 89 Springer LNCS #435.

[16] C.H.Bennett, F.Bessette, G.Brassard, L.Salvail, J.Smolin: Experimental quantum cryptography, Journal of Cryptology 5,1 (1992) 3-28.

[17] C.H.Bennett, G.Brassard, A.K.Ekert: Quantum cryptography, Scientific American (October 1992) 50-57.

[18] M.Ben-Or, S.Goldwasser, A.Wigderson: Completeness theorems for non-cryptographic fault-tolerant distributed computations, ACM Symposium on Theory of Computing STOC 20 (1988) 1-10.

[19] Daniel J. Bernstein: Multidigit multiplication for mathematicians, Adv. Applied Math. (2002?).

[20] J.Black, P.Rogaway, T.Shrimpton: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV, Crypto 2002 (Springer LNCS #????)

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

[22] Dan Boneh & Igor Shparlinski: On the Unpredictability of Bits of the Elliptic Curve Diffie–Hellman Scheme, pp.201-212 in Crypto 2001, Springer LNCS #2139.

[23] Raymond Bonner & Josh Barbanel: Democrats Rue Ballot Foul-Up in a 2nd County, New York Times, 17 Nov. 2000.

[24] Allan Borodin & Robert T. Moenck: Fast Modular Transforms, J. Comput. Syst. Sci. 8,3 (1974) 366-386. Older version (but not a subset) pp.90-96 in Moenck & Borodin: Fast modular transforms via division, in Richard M. Karp (chair): 13th annual symposium on switching and automata theory, IEEE Computer Society, Northridge, 1972.

[25] Fabrice Boudot: Efficient Proof that a Committed Number Lies in an Interval, pp.431-444 in Proc. of EuroCrypt 2000, Springer Verlag LNCS #1807.

[26] Fabrice Boudot, Berry Schoenmakers, Jacques Traoré: A fair and Efficient Solution to the Socialist Millionaires' Problem, Discrete Applied Mathematics 111 (July 2001) 23-36 (Special issue on Coding and Cryptology).

[27] J. Boyar, M.Krentel, S.Kurtz: A discrete logarithm implementation of perfect zero knowledge blobs, J.Cryptology 2,2 (1990) 63-76.

[28] G.Brassard, D.Chaum, C.Crépeau: Minimum disclosure proofs of knowledge, J.Computer System Sci. 37,2 (1988) 156-189.

[29] CALTECH/MIT VOTING TECHNOLOGY PROJECT: Voting machines and the underestimate of the Bush vote, report dated 9 Nov 2004.

[30] Jan L. Camenisch, Jean-Marc Piveteau, and Markus A. Stadler: Blind Signatures Based on the Discrete Logarithm Problem, EUROCRYPT '94, pages 428-432, Springer Verlag LNCS #950.

[31] Jan Camenisch & Markus Michels: Proving in zero knowledge that a number is the product of two safe primes, pp.107-122 in EuroCrypt 1999, Springer LNCS #1592.

[32] Peter Carlson: Another Race To the Finish, Washington Post Friday, 17 November 2000 (front page).

[33] Robert A. Caro, Means of Ascent (The Years of Lyndon Johnson, Volume 2) Alfred A. Knopf; Vintage (reprint edition 1991).

[34] David Chaum: Untraceable electronic mail, return addresses, and digital pseudonyms, Communications of the ACM 24,2 (Feb. 1981) 84-88.

[35] David Chaum: Secret-ballot receipts: true voter-verifiable elections, IEEE Security & Privacy 2,1 (Jan/Feb 2004) 38-47.

[36] Adam Cohen & Elizabeth Taylor: American Pharaoh: Mayor Richard J. Daley: His Battle for Chicago and the Nation, Little, Brown & Company 2000.

[37] Henri Cohen: A Course in Computational Algerbraic Number Theory, Springer (GTM #138) 1995.

[38] Preserving Democracy: What Went Wrong in Ohio, Status Report of the House Judiciary Committee Democratic Staff, 5 Jan. 2005.

[39] Ronald Cramer & Ivan B. Damgård: Zero-Knowledge Proofs for Finite Field Arithmetic or: Can Zero-Knowledge be for Free? 424-441 in Crypto 1998, Springer LNCS #1462.

[40] R. Cramer, I.Damgărd, U. Maurer: Secure and Efficient General Multiparty Computations from any Linear Secret Sharing Scheme, EUROCRYPT 00 (2000; Springer Verlag LNCS #???).

[41] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, Moti Yung: Multi-Authority Secret-Ballot Elections with Linear Work, pp.72-83 in EuroCrypt 1996, Springer LNCS #1070.

[42] R. Cramer, R. Gennaro, B. Schoenmakers: A secure and optimally efficient multi-authority election scheme, pp.103-118 in Advances in Cryptology EUROCRYPT 1997, Springer LNCS#1233. Journal version appeared in: European Transactions on Telecommunications 8 (Sept.-Oct. 1997) 481-490.

[43] John Brillhart, D. H. Lehmer, J. L. Selfridge, Bryant Tuckerman, S. S. Wagstaff, Jr.: Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers, American Mathematical Society, Providence RI, third edition, 2002. (The third edition is an electronic book available at `http://www.ams.org/online_bks/conm22`.)

[44] Joan Daemen & Vincent Rijmen: The Design of Rijndael: AES – The Advanced Encryption Standard Springer 2002.

[45] Ivan Damgård & Eiichiro Fujisaki: An Integer Commitment Scheme based on Groups with Hidden Order (Preliminary Version) http://eprint.iacr.org/2001/064 pp.125-142 in ASIACRYPT 2002, Springer LNCS #2501.

[46] Ivan Damgård, Jens Groth, Gorm Salomonsen: The Theory and Implementation of an Electronic Voting System , pp.77-100 in *Secure Electronic Voting*, (ed. D. Gritzalis) Kluwer Academic 2003.

[47] Ivan Damgård & Mads Jurik: A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System, Proc. of Public Key Cryptography 2001.

[48] Yvo G. Desmedt & Yair Frankel: Threshold cryptosystems, Crypto 89 (1990) 307-315 (Springer LNCS #435).

[49] Yvo G. Desmedt: Threshold cryptography, European Trans. Telecommunications 5,4 (1994) 449-457.

[50] Christophe Devine: Program to compute SHA-2/256 hash function: `http://www.cr0.net:8040/code/crypto/sha256/`.

[51] M. Dietzfelbinger: Primality Testing in Polynomial Time, Springer, 2004.

[52] A.DiFranco, A.Petro, E.Shear, V.Vladimirov: Small vote manipulations can swing elections, Communications of the ACM 47,10 (Oct.2004) 43-45.

[53] Netrinsics: Analysis of Precinct Return Data for Duval County, Florida `http://www.netrinsics.com/Duval/`.

[54] Netrinsics: Comparison of Precinct Return Data between Duval County and Lee County, Florida `http://www.netrinsics.com/DuvalVsLee/DuvalVsLee.html`.

[55] 77-page Edison-Mitofsky report on exit poll discrepancy 2004, available from their web site `http://www.exit-poll.net/`.

[56] Taher Elgamal: A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Transactions on Information Theory, IT-31,4 (1985) 469-472.

[57] Pal Erdös: Remarks on number theory III: addition chains, Acta Arithmetica 6 (1960) 77-81.

[58] S.Even, O.Goldreich, A.Lempel: A Randomized Protocol for Signing Contracts, Communications of the ACM 28,6 (1985) 637-647.

[59] R.Fagin, M.Naor, P.Winkler: Comparing Information Without Leaking It. Commun. ACM 39,5 (1996) 77-85.

[60] Barbara Silberdick Feinberg: American Political Scandals, Franklin Watts 1992.

[61] Amos Fiat & Adi Shamir: How to prove yourself: practical solutions to identification and signature problems. Proceedings of CRYPTO 86, Springer-Verlag 1987 (LNCS 263) 186-194.

[62] Federal Election Commission, `http://www.fec.gov/pages`.

[63] Bob Fitrakis, Steve Rosenfeld, Harvey Wasserman: Ohio GOP election officials ducking notices of deposition as Kerry enters stolen vote fray, Ohio Free Press 28 December 2004.

[64] Bob Fitrakis, Steve Rosenfeld, Harvey Wasserman: Ohio's Official Non-Recount Ends amidst New Evidence of Fraud, Theft and Judicial Contempt Mirrored in New Mexico Ohio Free Press, 31 December 2004.

[65] Steven F. Freeman: The Unexplained Exit Poll Discrepancy, part 1, manuscript dated 29 December 2004 and available from his web site `http://www.appliedresearch.us/sf/`. (According to Freeman, part 2 is to be published as a book.)

[66] G.Frey & H-W. Rück: A remark concerning -divisibility and the discrete logarithm in the divisor class group of curves, Mathematics of Computation 62 (1994) 865-874.

[67] A. Fujioka, T. Okamoto, K. Otha: A practical secret voting scheme for large scale elections, pp.244-251 in Advances in Cryptology - AusCrypt 92, Springer LNCS #718.

[68] John Fund: Stealing Elections: How Voter Fraud Threatens our Democracy, Encounter Books 2004.

[69] Jun Furukawa & Kazue Sako: An efficient scheme for proving a shuffle, 21st Advance in Cryptology Conf. Crypto (2001) 368-387. Springer (LNCS 2139). Jun Furukawa: Efficient and Verifiable Shuffling and Shuffle-Decryption, IEICE Trans. Fundamentals E88-A, 1 (Jan. 2005) 172-189.

[70] M.R. Garey & D.S. Johnson: Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[71] M.R. Garey, D.S. Johnson, L.J. Stockmeyer: Some Simplified NP-Complete Graph Problems, Theoretical Computer Science. 1,3 (1976) 237-267.

[72] R.Gennaro, S.Jarecki, H.Krawczyk, T.Rabin: Robust threshold DSS signatures, Information & Computation 164,1 (2001) 54-84.

[73] R.Gennaro, M.O.Rabin, T.Rabin: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography, Proc. ACM Symposium on Principles of Distributed Computing PODC 7 (1998) 101-111. Contains flawed secret-multiplication scheme. (The generalization by R.Cramer, I.Damgard, U.Maurer: Eurocrypt 2000 = springer LNCS #1807, 316-334 presumably has an analogous flaw.)

[74] Sam Giancana & Chuck Giancana: Double Cross: The Explosive, Inside Story of the Mobster Who Controlled America,

[75] Alan Gibbard: Manipulation of Schemes That Mix Voting with Chance, Econometrica 41 (1977) 587-600.

[76] O. Goldreich, S. Micali and A. Wigderson: How to Play Any Mental Game or a Completeness Theorem for Protocols with Honest Majority, Proc. 19th ACM Symposium on the Theory of Computing (STOC 1987) 218-229.

[77] Oded Goldreich, Silvio Micali, Avi Wigderson: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems, J.ACM 38,3 (1991) 690-728.

[78] Jens Groth: A verifiable secret shuffle of homomorphic encryptions, pp. 145-160 in Public Key Cryptography 2003, Springer-Verlag LNCS #2567.

[79] Jens Groth: Non-interactive Zero-Knowledge Arguments for Voting, pp. 467-482 in *Applied Cryptography and Network Security - ACNS 2005* (Springer LNCS #3531).

[80] Stuart Haber & W.Scott Stornetta: How to time-stamp a digital document, Crypto 90 (Springer LNCS #537) 437-455.

[81] J. Evetts Haley: A Texan Looks at Lyndon: A Study in Illegitimate Power, Palo Duro Press 1964.

[82] Lein Harn & Shoubao Yang: Public-Key Cryptosystem Based on the Discrete Logarithm Problem, pp. 469-476 in ASIACRYPT 1992.

[83] B. Harris: Probability distributions related to random mappings, Annals of Math'l Statistics 31 (1960) 1045-1063.

[84] Bev Harris: Black Box Voting – ballot tampering in the 21st century, Talion Publishing Co. 2004. ISBN=1-890916-90-0. Harris also maintains a web site about electronic voting at `www.BlackBoxVoting.org`.

[85] Peter Hellekalek & Stefan Wegenkittl: Empirical evidence concerning AES, ACM Transactions on Modeling and Computer Simulation 13,4 (2003) 322-333.

[86] Martin Hirt: Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting, PhD thesis, ETH Zurich, 2001. Reprint as vol. 3 of ETH Series in Information Security and Cryptography, ISBN 3-89649-747-2, Hartung-Gorre Verlag, Konstanz, 2001.

[87] Martin Hirt, Ueli Maurer, B.Przydatek: Efficient secure multiparty computation, Advances in Cryptology ASIACRYPT 2000 Springer (LNCS #1976) 143-161.

[88] Martin Hirt & Jesper Buus Nielsen: Upper bounds on the communication complexity of cryptographic multiparty communication, Cryptology ePrint Archive, Report 2004/318, Nov 2004, `http://eprint.iacr.org/`.

[89] Martin Hirt & Kazue Sako: Efficient receipt-free voting based on homomorphic encryption, Advances in Cryptology EUROCRYPT 2000, Springer (LNCS 1807) 539-556.

[90] John Hooper: Vote selling adds to German poll tensions, *The Guardian*, Monday 16 September, 2002.

[91] Markus Jakobsson & Ari Juels: Mix and match: secure function evaluation via ciphertexts, Asiacrypt (2000) 162-177, Springer (LNCS 1976).

[92] Markus Jakobsson, Ari Juels, Ronald L. Rivest: Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking, Proceedings of the 11th USENIX Security Symposium (2002) 339-353.

[93] Markus Jakobsson, Kazue Sako, Russell Impagliazzo: Designated verifier proofs and their applications, Eurocrypt (1996) 143-154, Springer (LNCS 1070).

[94] Rui Joaquim, André Zúquete, Paolo Ferreira: REVS – a robust electronic voting system, IADIS (Int'l Assoc. for Development of Information) Conference on e-society 2003.

[95] A. Joux & R. Lercier: Improvements to the general number field sieve for discrete logarithms in prime fields, Math. of Comput. 72,242 (2003) 953-967.

[96] Ari Juels, Dario Catalano, Markus Jakobsson: Coercion-resistant electronic elections, Cryptology ePrint Archive: Report 2002/165 `http://eprint.iacr.org/`.

[97] Adam Kalai: Generating random factored numbers, easily, Proceedings 13th ACM-SIAM Symposium on Discrete algorithms (6-8 Jan. 2000) 412 (1 page long) San Francisco, California. Also `http://people.cs.uchicago.edu/~kalai/factor/factor.html`.

[98] Dan Keating: Lost Votes in N.M. a Cautionary Tale Washington Post, Sunday, 22 August 2004, page A5.

[99] Aggelos Kiayias & Moti Yung: Self-tallying elections and perfect ballot secrecy, Public Key Cryptography 5 (2002) 141-158.

[100] Joe Kilian: Founding cryptography on oblivious transfer, ACM Symposium on Theory of Computing STOC 20 (1988) 20-31.

[101] J. Kilian, S. Micali, R. Ostrovsky. Minimum Resource Zero-knowledge Proofs, Proceedings, 30th Annual IEEE Symposium on the Foundations of Computer Science (FOCS 1989) 474-479. Also abstracted in Crypto 1989 pp. 545-546 (Springer LNCS #435). Also, these ideas are supposed to have been discussed in the following (but I have never seen it): Joan F. Boyar, Carsten Lund, René Peralta: On the Communication Complexity of Zero-Knowledge Proofs, J.Cryptology 6,2 (Spring 1993) 65-85.

[102] Joe Kilian & Erez Petrank: An Efficient Non-Interactive Zero-Knowledge Proof System for NP with General Assumptions, J. Cryptology 11,1 (Winter 1998) 1-27.

[103] George Knapp: Voter Registrations Possibly Trashed, KLAS TV, Tuesday 12 Oct. 2004. `http://www.klas-tv.com`.

[104] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin and Dan S. Wallach Analysis of an Electronic Voting System, IEEE Symposium on Security and Privacy, Oakland, CA, May, 2004; available at `http://avirubin.com`.

[105] John Kubiniec: Election fraud in Ukraine presidential vote, Freedom House press release Kyiv, Ukraine, 22 November 2004, `http://www.freedomhouse.org/media/pressrel/112204.htm`.

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

[107] Jonathan Levin & Barry Nalebuff: An introduction to vote-counting schemes, J. Economic Perspectives 9,1 (1995) 3-26.

[108] Helger Lipmaa: On Diophantine Complexity and Statistical Zero-Knowledge Arguments. pp.398-415 in ASIACRYPT 2003, Springer LNCS #2894.

[109] H.Lipmaa, N.Asokan, V.Niemi: Secure Vickrey Auctions without Threshold Trust, in *Financial Cryptography 2002* (Matt Blaze ed., Southampton Beach, Bermuda, 11-14 March 2002) Springer LNCS #2357. `http://www.tcs.hut.fi/ helger/papers/lan02/`

[110] W.J.M. Mackenzie: Free elections, George Allen & Unwin Ltd. London 1958.

[111] Rebecca Mercuri's Statement on Electronic Voting: `http://www.notablesoftware.com/RMstatement.html`, 2001.

[112] Daniel M. Merkle & Murray Edelman: A Review of the 1996 Voter New Service Exit Polls from a Total Survey Error Perspective, pp. 68-92 in *Election Polls, the News Media, and Democracy* (P.J. Lavrakas & M.W. Traugott eds.) Chatham, NY 2000.

[113] Dana Milbank: Washington Post, Sunday, 12 November 2000, page A22.

[114] Atsuko Miyaji: Elliptic Curves over F_p Suitable for Cryptosystems, pp. 479-491 in AUSCRYPT 1992.

[115] M.Naor & A.Shamir: Visual cryptography, pp. 1-12 in Euroocrypt 94, Springer (LNCS #950).

[116] C. Andrew Neff: Verifiable Mixing (Shuffling) of Elgamal Pairs, manuscript, `voteHere.net` 2004. Extends and corrects his earlier: A Verifiable Secret Shuffle and its Application to E-Voting. Proceedings ACM conf. Computer and Communications Security CCS-8 (2001) 116-125.

[117] Netrinsics statistical analysis of voting in 2000 in Duval and Lee Counties, Florida: `http://www.netrinsics.com/DuvalVsLee/DuvalVsLee.html` and `http://www.netrinsics.com/Duval/`

[118] V.Niemi & A.Renvall: Efficient voting with no selling of votes, Theoretical Computer Science 226, 1-2 (1999) 105-116.

[119] Kaissa Nyberg & Rainer A. Rueppel: Message recovery for signature schemes based on the discrete logarithm problem, pp.182-193 in Eurocrypt'94, Springer LNCS #950.

[120] One Last Election Lesson, New York Times, 18 January 2005, p. A20.

[121] T.Okamoto: Receipt-free electronic voting schemes for large scale electons, pp.25-35 in B.Christianson et al (ed) Security Protocols Workshop, Springer 1997 (LNCS 1361).

[122] Greg Palast: The best democracy money can buy, Pluto Press 2002.

[123] Tom Parfitt (Kiev) & Colin Freeman: Revealed: the full story of the Ukrainian election fraud, *The Daily Telegraph* 28 Nov. 2004.

[124] T.P.Pedersen: A threshold cryptosystem without a trusted party, Eurocrypt 91 (Springer LNCS #547) 522-526.

[125] Richard Hayes Phillips, Ph.D.: Hacking the vote in Miami County, Ohio Free Press 25 December 2004.

[126] Chellie Pingree (president of Common Cause): Voting Machine Makers Wine and Dine Officials, Scoop Independent News, 2 September 2004. `http://www.scoop.co.nz/stories/WO0409/S00046.htm`.

[127] S. Pohlig & M. Hellman: An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance, IEEE Transactions Info. Theory 24 (1978) 106-110.

[128] David Pointcheval & Jacques Stern: Provably Secure Blind Signature Schemes, ASIACRYPT '96 (Springer LNCS #1163) 252-265. New blind signatures equivalent to factorization (extended abstract) Proceedings of the 4th ACM conference on Computer and communications security, Zurich, Switzerland 1997, pages 92-99. Security proofs for signature schemes, EUROCRYPT '98 LNCS #1070 387-398 Security Arguments for Digital Signatures and Blind Signatures, J.Cryptology 13,3 (2000) 361-396.

[129] J.M.Pollard: Monte Carlo methods for index computation (mod p), Math'cs of Comput. 32,143 (1978) 918-924; Kangaroos, Monopoly and discrete logarithms, J.Cryptology 13 (2000) 437-447.

[130] Carl Pomerance (ed.): Cryptography and Computational Number Theory, Proc. Symp. Appl. Math., Volume 42, American Mathematical Society, Providence, RI, 1990.

[131] Michael Powell & Peter Slevin: Several Factors Contributed to 'Lost' Voters in Ohio, Washington Post, 15 December 2004 page A1.

[132] Michael O. Rabin & Jeffrey O. Shallit: Randomized Algorithms in Number Theory, Communications Pure & Applied Mathematics 39 (1986) 239-256.

[133] Thomas C. Reeves: A question of character, the life of John F. Kennedy, NY 1991.

[134] Arroyo Faces Renewed Attacks After Admission, REUTERS 28 June 2005.

[135] R.L.Rivest, A.Shamir, L.Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21,2 (Feb. 1978) 120-126.

[136] Bruce Schneier: Applied cryptography: Protocols, Algorithms, and Source Code in C, (2nd ed.) Wiley 1996.

[137] Berry Schoenmakers: A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting, Crypto 99 (1999; Springer LNCS #1666) 148-164.

[138] A. Schönhage & V. Strassen: Schnelle Multiplikation grosser Zahlen (Fast multiplication of large numbers), Computing, 7 911971) 281-292.

[139] René Schoof: Elliptic curves over finite fields and the computation of square roots mod p, Mathematics of Computation 44 (1985) 483-494. Also: Counting points on elliptic curves over finite fields, J.Théorie de Nombres Bordeaux 7 91995) 219-254. Both are available electronically with an errata sheet for the former at `http://www.mat.uniroma2.it/~schoof/papers.html`.

[140] René Schoof: Four primality testing algorithms, review article available at `http://www.mat.uniroma2.it/~schoof/papers.html`.

[141] J.T.Schwartz: Fast Probabilistic Algorithms for Verification of Polynomial Identities, J. ACM 27,4 (1980) 701-717.

[142] A.Shamir: How to share a secret, Commun.ACM 22,11 (1979) 612-613.

[143] Peter W. Shor: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comput. 26,5 (1997) 1484-1509.

[144] Adi Shamir: IP=PSPACE, Journal of the ACM 39,4 (1992) 869-877.

[145] A.Shen: IP=PSPACE, simplified proof, J.ACM 39,4 (1992) 878-880.

[146] Victor Shoup: Lower bounds for discrete logarithms and related problems, pp.313-328 in EuroCrypt 1997, Springer LNCS #1233.

[147] Bill Sloat: Judge Orders Halt to County Hearings Challenging Voters, Cleveland Plain dealer, 30 Oct. 2004, page A1.

[148] Nigel P. Smart: The discrete logarithm problem on ellitpic curves of trace one, J.Cryptology 12,3 (1999) 193-196.

[149] Warren D. Smith: Range voting, #56 at `http://math.temple.edu/~wds/homepage/works.html`.

[150] Warren D. Smith: Reweighted Range Voting – new multiwinner election scheme, #78 at `http://math.temple.edu/~wds/homepage/works.html`.

[151] Warren D. Smith: "Asset voting" scheme for multiwinner elections, #77 at http://math.temple.edu/∼wds/homepage/works.html.

[152] Warren D. Smith: New cryptographic voting scheme with best-known theoretical properties, #89 at http://math.temple.edu/∼wds/homepage/works.html.

[153] Warren D. Smith: Manuscript work in progress on sar-range voting, 2004.

[154] Jerome A. Solinas: Efficient arithmetic on Koblitz curves, Designs Codes & Cryptography 19,2-3 (2000) 195-249.

[155] Larry J. Stockmeyer: Planar 3-colorability is NP-complete, SIGACT News 5,3 (1973) 19-25. Stockmeyer's reduction is also briefly described in the easier-to-obtain Jörg Rothe: Heuristics versus completeness for graph coloring, Chicago J. Theoretical CS (2000) article 1.

[156] Tom Stuckey: Report: Maryland voting system vulnerable to hackers, *Associated Press* 1 Feb 2004.

[157] D. Stucki, N. Gisin, O. Guinnard, G. Ribordy, H. Zbinden: Quantum Key Distribution over 67 km with a plug & play system, quant-ph/0203118. Submitted to the New Journal of Physics.

[158] Swarthmore college students' posting of leaked Diebold emails: http://scdc.sccs.swarthmore.edu/diebold/lists/support.w3archive/200101/msg00068.html

[159] Edlyn Teske: On random walks for Pollard's rho method, Mathematics of Computation 70,234 (Apr. 2001) 809-825.

[160] Ken Thompson: Reflections on trusting trust (Turing Award lecture), Communications of the ACM 27, 8 (August 1984) 761-763.

[161] Nicolaus Tideman: The Single transferable Vote, J. Economic Perspectives 9,1 (1995) 27-38. (Special issue on voting methods.)

[162] P.C.van Oorschot & M.J.Wiener: Parallel collision search with cryptanalytic applications, J.Cryptology 12,1 (1999) 1-28.

[163] W. Vickrey: Counterspeculation, Auctions, and Competitive Sealed Tenders, Journal of Finance (March 1961) 8-37.

[164] X.Wang, Y.L.Yin, H.Yu: Collision search attacks on SHA-1, Shandong University, China, preprint.

[165] C. Zalka: Grover's quantum searching algorithm is optimal, Physical Review A, 60 (1999) 2746-2751. Also quant-ph/9711070.

[166] Tom Zeller, Jr.: Vote Fraud Theories, Spread by Blogs, Are Quickly Buried, New York Times (Front page) 12 Nov 2004.

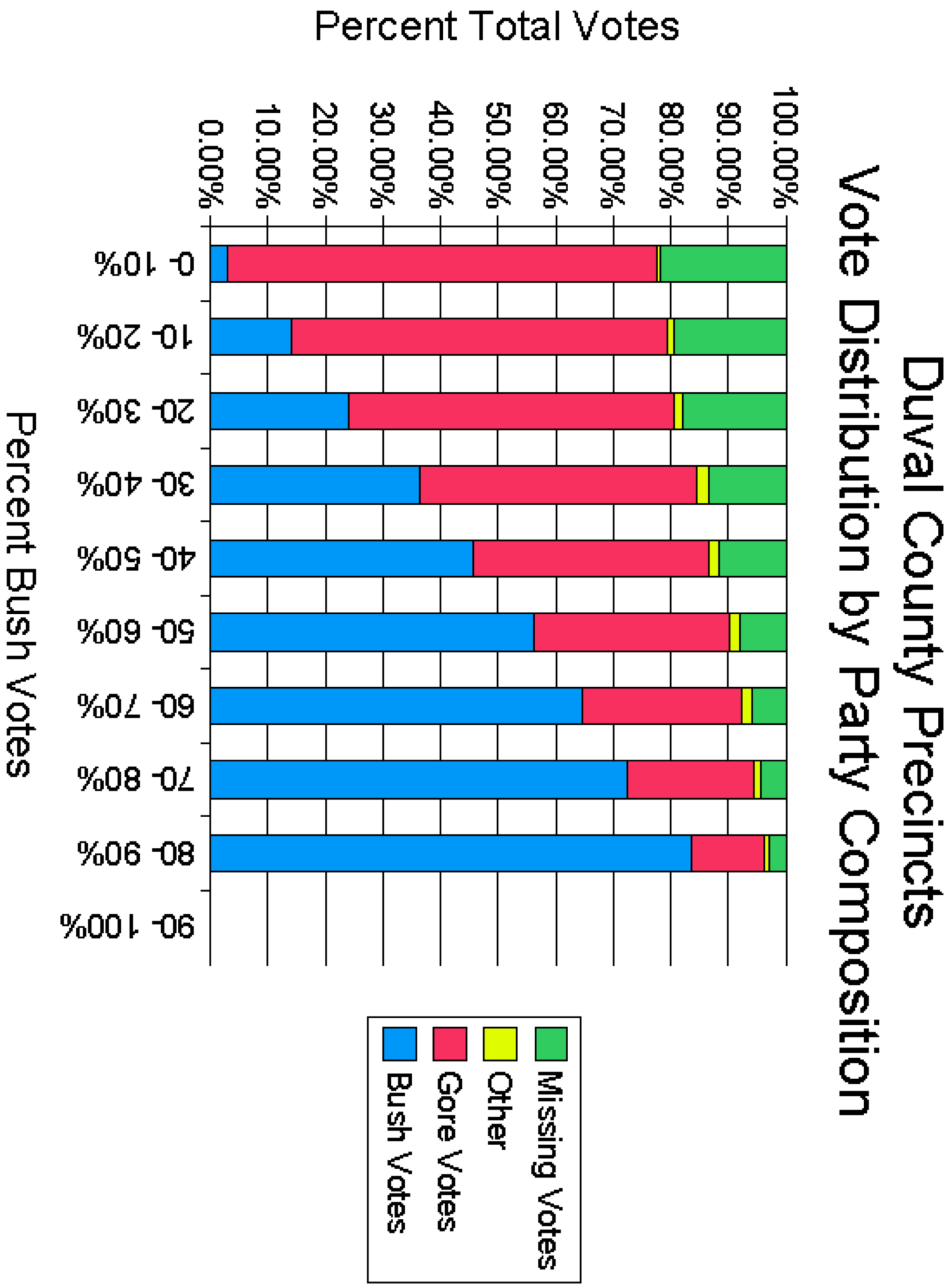

Figure 11.1. [117] How the 268 precincts in Duval County, Florida, voted in the 2000 Bush-Gore presidential race. Observe the remarkably linear negative-slope fit between the percentage of missing votes in that precinct (ruled invalid by the election authority and not counted) and its percentage of votes for Bush. ▲