New cryptographic election protocol with best-known theoretical properties

Warren D. Smith
warren.wds@gmail.com

September 9, 2005

Abstract — **We describe a correct, "verifiable," and "coercion resistant" election scheme which takes $O(N+V)$ (highly parallelizable) steps to process V votes by N voters. It can handle essentially any underlying vote-combining method, provided the number of possible distinct votes is far smaller than the number of voters. In these theoretical senses (aside from the proviso) the new voting protocol is optimal.**

Juels, Catalano, and Jakobsson had introduced the first such scheme in 2002, except that it consumed quadratic work and communication bandwidth and was vulnerable to two attacks (described here). Our scheme is obtained from JCJ's by first modifying it to make it immune to the two attacks, then adding two ideas to speed it up to linear time, then finally adding a third idea to reduce the constant factor in the O to the point where it apparently becomes practically feasible.

The extra security guarantees encapsulated by "coercion resistance" perhaps are not enough, in a practical sense, to make our new protocol more desirable than simpler schemes based on homomorphic encryption and bulletin boards. That is because our new scheme makes heavy use of mixnets and cooperative computation on "shared secrets" carried out by mutually distrustful parties – both of which cause our communication and/or verification requirements to be comparatively large.

Finally, we remind the reader that still *no* secure voting scheme is presently known that is anywhere close to feasibility, for handling *non*-additive election methods that involve enough bits per vote to allow typical voters to generate *unique* votes. The author's "reweighted range voting" is an example of such an election method – it arguably is the best one available for multiwinner elections – and the cryptographic community so far has not even considered how to handle such elections.

1 Cast of characters

Bulletin boards (BBs): Memory which may be read (with random access) by anybody, and which may be written by approved agencies. It is usually convenient to assume that this writing is always of "append" (rather than "random access") style.

Voters: Provide votes (in encrypted form) which are posted on a bulletin board.

Mixers: Accept N encrypted inputs, permute and re-encrypt those inputs, and output the N results along with ZK-proofs[1] that they did so. Several mutually distrustful mixers, one after the other, can thus perform a permutation and simultaneous re-encryption of the data with *nobody* knowing what the product permutation is. (This is called a "mixnet." It is important that the mixers distrust one another because that way at least some will refuse to collude and will not tell each other their secret permutations.)[2]

Talliers: There are several, mutually distrustful, talliers. Each tallier knows some secret information not known to the other talliers. This allows super-threshold subsets of the talliers to perform computations cooperatively that would be infeasible for any subthreshold set. (It is important that they distrust one another[3] because they will refuse to collude and will not tell each other their secrets.)

Verifiers: The voters and talliers and mixers broadcast enough information in the form of "zero knowledge proofs" (preferably "non-interactive" ones[4]) to permit any external verifier, by examining that information, to become confident that the talliers, voters, and mixers are performing the computations they are supposed to (or confident that some – whose identities will be apparent or irrelevant – are cheating, or that super-threshold sets of cheating talliers and/or mixers are colluding).

2 Informal definitions of Security guarantees

Correctness: ① Each authorized voter's chronologically-last-cast[5] vote is incorporated – correctly – into the computed election result, but ② no unauthorized voter can have his vote counted and ③ no voter can have more than one vote counted.

Verifiable: After the election, the facts that ①-③ were true are proven by a zero-knowledge proof that is available to external verifiers – or (if one or more were false) then the ZK-proof protocol makes it clear which actor first violated the protocol.

[1] ZK is an abbreviation for "zero knowledge."
[2] It is simplest conceptually to regard the mixers and talliers as disjoint entities. However, they could in fact be the same entities.
[3] Unless the election is a total sham, at least some of the competing *candidates* ought to distrust each other.
[4] To reduce communication needs, and also to allow creation of a permanent record, reviewable at a later time, of the election.
[5] Other vote-selection conventions could also be considered.

Coercion resistant: "We allow the adversary to demand of coerced voters that they vote in a particular manner, abstain from voting, or even disclose their secret keys. We define a scheme to be coercion-resistant if it is infeasible for the adversary to determine whether a coerced voter complies with the demands." [19]

Our schemes shall depend on the assumed difficulty of the discrete logarithm problem in elliptic curve groups of large prime order.

3 Sketch of the JCJ scheme [19]

The identity of voters remain hidden throughout the JCJ voting process. Each ballot contains inside it a "concealed credential" in the form of an encryption C of a secret value σ known only to that voter, and an encryption B of the vote itself, and a NIZK (non-interactive zero knowledge) validity proof P of that vote[6] – and P also includes a NIZK proof of knowledge of σ indecomposably ANDed (§A.3) with the first proof:

$$(C, B, P). \tag{1}$$

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 ① the hidden credentials on the (scrambled-order and re-encrypted) list of votes, and ② 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 via a "plaintext equality test" (PET, see §A.7) on data that has been encrypted via randomized ElGamal[7] encryption. Of course the tallier broadcasts ZK validity proofs of all the mixing and plaintext-equality-testing.

This allows the tallier to determine each vote's legality, and to prevent double voting, but without knowing any vote's author or content, and indeed without it ever being revealed *who* voted.[8] If several votes by the same voter are detected by PET 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, posted, and totaled.

JCJ employ several mutually distrustful talliers who cooperatively decrypt [9][23][10][11] 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 super-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. Because the votes are encrypted, and the decryption key is known only in a shared-secret sense to the talliers, but never known to any individual (nor indeed even to any subthreshold set of talliers) nobody can decrypt the votes, until, at the very end, the talliers do so cooperatively (and post the resulting plaintext votes on a BB).

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 (or encryption, after several re-encryptions are done[9]) key, and the encryption method is ElGamal randomized.

4 Criticism of JCJ scheme

The JCJ scheme has, in most respects, better security guarantees than any other voting scheme I know, and indeed JCJ formulate a rigorous definition of a strong property they call "coercion resistance" and claim their scheme satisfies it. Nevertheless it has the following three weaknesses:[10]

[6] I.e. that it represents the name of an actual candidate, if we are speaking of a plurality-type election.

[7] JCJ actually propose using *modified* ElGamal encryption $(f^r, g^r, h^r M)$ of a message M where $h = f^k g^\ell$ and the fixed random group elements f, g, h are public while the discrete logarithms k and ℓ are secret. We for simplicity here are considering only the original ElGamal scheme (i.e. with $f = 1$) but nothing prevents us from changing everything to modified ElGamal throughout if we desire to follow JCJ more closely.

[8] While JCJ consider this to be a benefit, others might consider it to be a bad thing. *Benefit*: If nobody can tell whether I voted, then supposedly nobody can coerce or bribe me not to vote. (This in fact is essential to prevent forced-abstention attacks. If there are 1000 voters, an adversary could coerce 200 of them not to vote, then check there were exactly 800 votes... but that attack would not work if only 449 among those 800 actually voted, as in the contemporary USA. [This attack was suggested by R.S.Araujo.]) *Demerit*: If I cannot prove to anybody that I did not vote, then an enemy might physically prevent me from voting, and then I would be unable to prove to possible rescuers that I had been prevented. In BB-homo election schemes (§9) each vote is digitally-signed by its author and posted on a BB under his name, so that everybody knows who voted. Therefore BB-homo schemes allow voters to prove they did or did not validly vote, but (consequently) also allow "forced abstention" attacks by coercers. (Note: In Australia, voting is mandatory and non-voters must pay a fine. The JCJ scheme would therefore be unsuitable for use in Australia, but BB-homo schemes – which reveal who voted – would be Australia-compatible.) We could with JCJ still provide *designated-verifier* ZK-proofs to each voter that their vote was used (§A.4), and still allow voters to prove or disprove that they successfully *pre-registered*. Thus voters would immediately *know* if their vote had been discarded (or not) but would be unable to prove that to anyone else. This seems a considerable demerit - because I consider schemes based on discarding a percentage of votes from certain geographical areas to be a common manipulation in the contemporary USA, but consider vote-coercion and vote-selling to be much less common. JCJ is vulnerable to an Election Authority that decides to discard 10% of all input-votes from all Florida Counties with large Black populations. *Rejoinder*: if each voter votes several times, then that attack would be much less effective. Also, if each voter submits votes to *several* EAs including *his favorite* EA (and there are several mutually distrustful EAs) then his vote is unlikely to be discarded by them all. Also, each time an EA discards a vote, the voter would immediately *know* it because he would not see data appending to the BB with a designated-verifier ZK-proof that it was his vote. That might inspire that voter to try another EA. [In BB-homo schemes where each vote is posted under that voter's name, a corrupt EA/BB could simply refuse to post the votes of voters it had pre-decided to block. Those voters would then be able to prove they had not successfully voted, and could *try* to get help.]

[9] More precisely, no individual is able to re-produce the encryption and thus demonstrate to a coercer that a specified vote is present.

[10] Note: when I wrote this, I was working from notes I had made some years before about the (then) version of [19]. But the newer version of [19] now available on Ari Juels' web page, now is mostly immune to the attacks we give, apparently because, during the intervening years, Juels et al thought of almost exactly the same attacks I did and then also thought of almost exactly the same defenses I did. We shall nevertheless leave

"1009 attack": The coercer could force his voter victim to provide a large number, say 1009, of identical coerced-votes, and then later see if *any* legitimate voter provided exactly 1009 identical votes – a fact that would be apparent to everybody since public plaintext-equality proofs had been broadcast to justify every elimination of a duplicate vote. (If no valid vote was duplicated exactly 1009 times, the coercer concludes that his voter victim either provided a fake credential, or provided additional uncoerced votes; either way, the coercer now has reason to punish his victim.)[11]

"Timestamping attack": If votes also contain a date and time so that the chronologically last-cast identically-credentialed vote may be kept, doesn't that allow voters to effectively uniquify their vote, and thus (by remembering the exact time) sell their vote?[12]

Unusable with uniquifiable votes: The JCJ scheme is unusable[13] with voting systems such as reweighted range voting [27] in which it is feasible for voters to uniquify their votes, with high probability distinguishing their own vote from all others.

The **main defect** of the JCJ scheme is that processing V votes by N voters requires $O(NV)$ steps to perform all the cross checks, i.e. at least N^2 steps. With 10^8 voters, some multiple c of 10^{16} group-exponentiations is required, which, even assuming c exponentiations could be done in 1 millisecond, would require 3000 centuries of compute-time. With present day prices and technology, this amount of computing seems unacceptable, although perhaps not impossible.

What *really* makes this absolutely unacceptable is the huge number (of order NV) of proofs of failed plaintext-equality-tests that need to be provided to all verifiers to justify keeping the non-bogus votes; that amount of storage, communication, and verification work is absolutely unacceptably huge.

5 Two (and a half) alterations of JCJ designed to defeat the "1009" and "timestamping" attacks

A "1009 fix" that does not work: To defeat the 1009 attack, we could simply impose some fairly small upper limit L (e.g. $L = 10$) on the number of times one is allowed to vote.

While at first this seems a good idea (and it would also have the benefit of bounding the runtime) it actually is not. The problem is that this would allow a coercer to prevent a voter from voting by forcing him to submit $\geq L$ coerced votes. If you submit L valid votes then I have forced you to vote the way I want; if you submit bogus votes then I have deprived you of the right to vote.

First successful altered version: Suppose we change JCJ so that

1. *All* duplicate votes are discarded (i.e. if there are M identically-credentialed votes, then all M are discarded, not just $M - 1$).
2. This is done *before* comparison with the official credential list, by self-comparison of the vote-list.

Result: the altered algorithm now runs in V^2 time to process V votes, which unfortunately is slower than the original $O(VN)$ runtime bound.

But it now seems immune to "1009 attack": If I force you to provide 1009 cloned votes, this is not useful to me, since all 1009 will be discarded, and I have no way to know if any had a valid credential because they are discarded before comparison with the official credentials.

This version also is immune to the "timestamping attack" because since all duplicate votes are discarded, there is no need to have any timestamps since there is no need to find the chronologically last duplicate.

This altered-JCJ suffers from the disadvantage that you have to be "sure" when you validly-vote, since you cannot correct a mistake by making a later re-vote.

Second successful altered version:

1. Mix and encrypt votes.
2. By self-comparison of vote-list credentials via plaintext equality testing, remove all but 1 of each equivalence-class of identically-credentialed votes. This one could (optionally) be the chronologically last.
3. Remove timestamps from votes.
4. Re-mix and re-encrypt the resulting pruned and timestamp-free vote list.
5. Compare votes via P.E.T. to (mixed & encrypted) official credential list; remove bogus votes.
6. Decrypt, post, and then count the votes.

This altered algorithm also requires V^2 time to process V votes, again slower than the original $O(VN)$ runtime bound.

our discussion as is, because it is informative to go through the attack and defense thought processes, and also because there are some differences in our solutions.

[11] I admit this is not that strong an attack, since there could be two coercers who both thought of the same number "1009"..., and also the coercer cannot know for *whom* those 1009 votes were cast, unless he watches the voter voting. If I force you to vote 1009 times for Hitler (and watch you do it), then you could try to defeat this coercion by using a fake credential, then remember "1009," then later vote a second time 1009 times for Gandhi, but now using a genuine credential. But I could defeat that by imprisoning you and preventing you from voting a second time – or giving you a drug that prevented you from remembering "1009". But there would be no need for such extreme measures if you *wanted* to *sell* your Hitler-vote to me, and were willing to provide reasonably convincing evidence to me that you had not re-voted at any other time. So this attack undermines the no-sale/no-coercion guarantees JCJ claim to provide, to at least some extent.

[12] JCJ could defeat this attack simply by not having any timestamping, and choosing *randomly* the vote to keep from a set of identically-credentialed votes. Then timestamping and last-cast-choosing could still be effectively got as follows: each time the voter changes his mind and re-votes, he submits 9 times more votes than all the previous votes he cast. That way, a randomly chosen vote will be of the chronologically last-cast type, with probability 90%. Unfortunately the cost of this idea is tremendous: after m mind-changes, a voter will have cast $(9^{m+1} - 1)/8$ votes!

[13] More precisely, it is still useable, but it loses all its coercion-resistance. Reweighted range votes contain "real numbers" having a large number of bits inside each vote; they are not just one bit per vote. Therefore a voter can point to his unique vote in the final decrypted vote list and thus sell it or prove its presence to a coercer.

But this method now *permits* correcting a mistake by making a later re-vote, and still is immune to the "1009 attack."[14] It also is immune to the "timestamping attack."

The L-limit fix revisited: Although we began by disparaging the "L-limit fix," it *would* work if there were no limit on the number of votes you could cast – there only was a limit L on the number that you could cast that had any particular credential. (Any credential used more than L times would, we agree, be eliminated from the election along with all votes that employed it.) Then any coercer would be unable to tell that a voter had simply employed an invalid credential with his coerced "votes." *But* this "revised L-limit fix" would have to be used *in combination* with one of our two successful altered schemes above and like them would require $O(V^2)$ steps to eliminate the too-duplicated votes via PET self-comparison of the credentials in the V-vote list.

Unfortunately the originally-hoped-for "benefit" of the L-limit idea – of keeping V small to speed it up – does not materialize with this revised L-limit idea.

Why all these alterations still are inadequate: Although both these alterations defeat the "1009" and "timestamping attacks," they make an already unacceptably slow voting technique become even slower.

Let V be the number of submitted votes, including duplicates and/or bogus votes and let N be the number of voters. Then V can be much larger than N.

So our new runtime V^2 can be much larger than N^2 and NV, which already are unacceptably large compared to what it ought to be, namely $O(V + N)$ time.

Runtimes that grow significantly faster than $O(V+N)$ permit **"denial of service attacks"** where the attacker submits an enormous number of bogus votes. Thus V becomes very large. Although these votes will not affect the election result, they will force JCJ's scheme (and even more so, our slower altered-JCJ schemes) to run so slowly that the election effectively will be cancelled.

The attacker can do this even with far smaller computational and communication resources than the election authority, since the attacker's runtime is $O(V)$ while the EA's (election authority's) is $O(V^2)$.

Our goal: Here is a quote from JCJ [19]:

> A drawback of our scheme is that, even with use of asymptotically efficient mix networks... the overhead for tallying authorities is *quadratic* in the number of voters. Thus the [JCJ] scheme is only practical for small elections. Our hope and belief, however, is that our proposed scheme might serve as the basis for refinements with a higher degree of practical application.

This paper provides exactly such refinements.

6 First idea behind new voting scheme – speedup to linear time

The primary problem with JCJ was its superlinear runtime. It would be better to devise a JCJ-like scheme running in only $O(V+N)$ time, or nearly so, e.g. $O(N \log N + V \log V)$ steps.

To accomplish that, we need to get rid of the "plaintext equality tests." Because PETs can only be used to spot all equalities by all-pairs testing, their use forces order NV runtime to compare a V-element list of votes to an N-element list of credentials, and order V^2 runtime to eliminate duplicates by self-comparison of a V-element vote list.

If we instead could eliminate duplicates by sorting and merging, the $O(NV)$ runtime would be reduced to $O(N \log N + V \log V)$. By instead sorting and binary searching, it would be reduced to $O(N \log N + V \log N)$. An even better approach would be to eliminate duplicates by using lookup in a hash table [20]; then (under randomness assumptions) expected runtime would be reduced to $O(N + V)$.

This is a fine plan, but it is not immediately clear it is possible without violating vote-privacy and anti-coercion guarantees. For example, if each voter simply encrypts his credential with some fixed deterministic encryption scheme, then hashing becomes possible *but* the voter can prove (by visibly re-encrypting) his vote is in the final list, and thereby sell his vote. If each voter encrypts using a randomized encryption scheme (possibly with more randomized re-encryption – unknown to and uncontrollable by the voter – during mixings later) then maybe the voter will not be able to sell his vote, but hashing by credential now seems impossible since each credential has 10^{300} possible randomized-encrypted forms.

7 Second idea behind new voting scheme – "secret encryptions"

To allow the speedup idea to proceed, we devise a new technique: "secret encryptions." These are encryptions for which nobody knows either the encryption[15] or decryption key. It is possible for several parties to perform such an encryption cooperatively by using "secure general purpose multiparty computation" (SGMPC)[16] on "shared-secret" data and using an "immaculately conceived shared-secret" encryption key.

Definitions: A "shared secret" is data that is collectively known to a set of sharers – any threshold-cardinality subset of whom could recover it, after performing enough cooperative computation – but no subthreshold set can recover the secret, or even any partial information about it, no matter how much computation they do with that part of the data that they possess. An "immaculately conceived shared-secret" is a shared secret which is generated cooperatively in such a way that no individual ever knows the secret.[17] Standard secret

[14] If I force you to provide 1009 cloned votes, this is not useful, since 1008 will be discarded, and I have no way to know if the remaining 1 vote had a valid credential because the 1008 are discarded before comparison with the official credentials, and then a re-mix and re-encryption occurs, so that finally, the remaining 1 vote might or might not have a valid credential – there is no way for me to tell or to tell which vote it is.

[15] More precisely, nobody knows the encryption process including all random bits used during that process, so that nobody can feasibly re-do it to regenerate a given encryption from a given plaintext.

[16] Or: just iterated encryption (with broadcast ZK-validity proofs for each one) could also work.

[17] The power of immaculate secrets is that they can encode important information, but no individual can know (and hence misuse) that information.

sharing schemes [25][12] provide immaculate conception capability, in addition to secret-share-deal-out, secret-recovery, and cheater-detection protocols.

All secret-sharing schemes work, and prevent cheating, *provided* that at least a threshold-cardinality subset of the sharers obey the protocol.

8 The protocol

Here is the voting scheme that results from grafting the above two improvement ideas onto JCJ's scheme. (Warning: this works, but only, as we shall see in §10, at immense cost. We shall provide a third idea in §11 to reduce the cost.)

procedure voting-scheme

1: **[Set-up]** Talliers and mixers generate the $O(1)$ immaculate shared secrets (and derived quantities, such as broadcast public keys) that will be used in the remainder of the protocol:

immaculate shared secret:	D_2, D_4	D_1	K	J
derived public quantity:	K_1	K_0		
used in step #:	4, 9, 16	15	9	15, 16

2: Initially: we *assume* a bulletin board BB_1 exists that exhibits the list of registered voters (names and addresses)[18] each with his (randomized ElGamal) encrypted credential C. The encryption key K_0 is publicly known but the corresponding decryption key is an immaculate shared secret. Also the randomness used to perform each encryption is an immaculate shared secret.[19]

3: **[Voting]** Election begins.

4: Voters submit tuples (B, P, T, C) to a bulletin board BB_2, where B is their encrypted ballot, P is a zero-knowledge validity proof for that ballot, T is the time the ballot was cast, and C is a (randomized ElGamal) encryption of that voter's credential (with different randomness from in step 2 of course). Each ElGamal encryption of C is done using a common publicly known encryption key K_1; the corresponding decryption key is an immaculate shared secret. (Voters can vote repeatedly, but only their last-cast valid vote ultimately will be counted.)[20]

5: Verifiers meanwhile confirm that the votes on BB_2 did appear only in an append-only manner and only with strictly monotonically increasing timestamps. Otherwise BB_2 is revealed as cheating.

6: The entries on BB_2 with invalid ZK-validity proofs P are marked "bogus" and will not be used in what follows.[21] (This would best be done interactively with the voting, so that voters would immediately be notified they had a bogus validity proof and therefore could try again.) All P-datafields may be discarded in what follows.

7: **[Anonymization of votes & elimination of duplicate or unauthorized votes]** The V non-bogus entries of BB_2 are run through a "mixnet." The output is a shuffling and re-encryption of the entries on BB_2. The shuffling permutation is known to nobody. The encryption key is publicly known but the decryption function D_2 is known to nobody (it is an immaculate shared secret).

8: Each of the V outputs of this mixnet are dealt out to the talliers (using the secret-sharing deal-out protocol). Call the resulting list of shared secrets L_2.

9: For each of the shared-secret entries $y \in L_2$, the talliers cooperatively compute $Z(y)$ where Z is a function which, given an encrypted (B, T, C)-tuple, outputs a (re)encryption of B, the decrypted plaintext of T, and $\sigma' = F_K(D_2(C))$ where $D_2(C) = \sigma$ is the decrypted plaintext credential σ corresponding to C, and $F_K(\cdot)$ is the AES[22] encryption function with immaculate-shared-secret key K. (Note: the talliers never know σ except in shared-secret or encrypted forms.) They post this output publicly on BB_3.

10: By hashing the σ', remove all but the last-posted among the BB_3 entries with duplicate credentials σ. Then also remove the time-stamp datafields T from all entries. The result is posted as BB_4.

11: The V' entries of BB_4 (where $V' \le V$) are run through a different mixnet. The output is a shuffling and re-encryption of the entries on BB_4. The shuffling permutation is known to nobody. The encryption key is publicly known but the decryption function D_4 is known to nobody (it is an immaculate shared secret).

12: Each of the V' outputs of this mixnet are dealt out to the talliers. Call the resulting list of shared secrets L_4.

13: The N entries of BB_1 are run through a different mixnet. The output is a shuffling and re-encryption of the entries on BB_1. The shuffling permutation is known to nobody. The encryption key is publicly known but the decryption function D_1 is known to nobody (it is an immaculate shared secret).

14: Each of the N outputs of this mixnet are dealt out to the talliers. Call the resulting list of shared secrets L_1.

15: For each of the shared-secret entries $x \in L_1$, the talliers cooperatively compute $F_J(D_1(x))$ where F_J again is the AES secret key encryption function, but now with key J

[18] BB_1 would have to be created during a previous "registration" phase. We intentionally have not discussed that phase because we have nothing new to say about it.

[19] To accomplish this, the key steps in creating BB_1 could include the following. Joe Voter transmits an ElGamal encryption of his credential σ to the EAs. They then re-encrypt it using immaculate-shared-secret randomness, transmitting a designated-verifier ZK-proof (;A.4) that they did so, back to Joe; the result is posted to BB_1 under Joe's name and with Joe's signature confirming he agrees the post is valid. Nothing stops corrupt EAs from refusing to register Joe, or from registering enormous numbers of nonexistent "voters." However, if the registered-voter list is *available for public view* for months before it is used, then independent journalists, etc. could check a random sample of the list to become confident it had a low (or high) degree of fakery.

[20] A voter who sees his vote has not appeared on BB_2 can try to re-cast his vote from different vote-collection centers and different EAs until it does appear!

[21] Attempts to submit extra fraudulent votes by bitwise-copying somebody's previously submitted C and re-using it with a *different* B, are defeated because each validity proof P is an *indecomposable* AND of an NIZK proof of knowledge of the plaintext form σ of C, and ballot-validity NIZK proof for B. For extra confidence we could also demand that P include an NIZK proof of knowledge of the plaintext form of B. We could also demand that P also incorporate (all "incorporations" via a secure hash function) bits from the timestamp T into its internal "challenges."

[22] AES is the USA's *a*dvanced *e*ncryption *s*tandard – or we could use any comparable secret-key cryptosystem or a "cryptographic hash function" got by erasing some AES output bits.

(or merely a secure hash function, which is essentially the same thing but with an agreed subset of its output bits erased). They post this output publicly on BB_5.

16: For each of the shared-secret entries $y \in L_4$, the talliers cooperatively compute $Q(y)$ where Q is a function which, given an encrypted (B, C)-tuple, outputs a (re)encryption of B together with $\sigma'' = F_J(D_4(C))$. They post these σ''s publicly on BB_6.

17: By hashing, delete those entries on BB_5 whose encrypted credential values σ'' are not already present on BB_6. The resulting list of validly-credentialed votes is posted on BB_7. This whole step is carried out fully publicly.

18: Each of the $N' < N$ entries on BB_7 are dealt out to the talliers, after removing their C-datafields which will no longer be needed. Call the resulting list of shared secrets L_7.

19: **[Final tallying]** By cooperative computation using the immaculately-shared decryption keys, the talliers compute, for each entry $u \in L_7$, the full decrypted ballot B inside that u, and posts it on BB_8.

20: Anybody can now trivially compute the election result from BB_8.

Obviously, the runtime of this scheme, with any bounded number of talliers and mixers, is $O(V + N)$ steps. Each "step" takes constant time – but it could be a very large constant because steps 9, 15, 16, 19 each involve using secure general-purpose multiparty computation (SGMPC, §A.12) to simulate a circuit that computes the composition of an ElGamal decryption and an AES encryption. That is a complicated boolean circuit and converting a boolean circuit to a secure multiparty computation on shared-secret input and output data involves a large – but bounded! – slowdown factor.

9 How good is it (theoretically speaking)?

Security guarantees: Our voting scheme appears to be *correct*, *verifiable*, and *coercion-resistant*. We have not provided formal proofs of these claims but instead – because our scheme may be regarded as merely a refinement/speedup of the JCJ scheme – rely on their reasoning. We should have at least as much security as JCJ since basing everything on our altered versions of JCJ gives us immunity to the 1009- and timestamping-attacks. These JCJ security guarantees seem stronger than any other proposed set of voting security guarantees in the literature. We gave [19]'s informal definition of "coercion-resistance" in §2 (and they also gave a formal definition in an appendix).

Speed: The total work is $O(V+N)$ to process V votes cast by N eligible voters. This is best possible. Unfortunately "secure general multiparty computation" (SGMPC) is employed as a "big gun" to overcome obstacles. This causes the constants hidden in our "O" to be large. Fortunately SGMPC is only used for small computations (each one involving only a single vote and/or a single small shared secret) each of which can be done independently from all others and in parallel.

Versus BB-homo: Compare our JCJ-based scheme with my other favorite secure voting method [6][7], which is based on voters posting their encrypted and ZK-validity-proved votes to their own name's entry on a bulletin board, then adding up the votes in encrypted form via homomorphic encryption, then finally a small cooperative computation decrypts the total. Both schemes run in optimal $O(V + N)$ work (and both schemes permit heavy parallelization of that work). Our JCJ-based scheme has better theoretical security guarantees than BB-homo. But that security comes at a heavy price:

1. **Storage and attacks based on storage limits:** The BB-homo scheme requires only $O(N)$ storage, which is better than our $O(V + N)$ storage needs. This permits an attacker to try to submit an enormous number of votes and thus overwhelm our storage capacity and effectively cancel the election. The BB-homo scheme is immune to such an attack. In practice this probably would not matter because, if votes were submitted by actual people in voting booths, then at most, say, 1 vote could be cast per voting booth per minute, not permitting V to become too large.
2. **Communication:** Mixnets inherently require a great deal of communication between distrustful mixers, and between mixers and verifiers. Our JCJ-based scheme not only employs mixnets, but also giant public hash tables and also cooperative computations performed *on each vote*. That all adds up to *a lot*[23] *of communication* both between distrustful talliers and mixers and between them and verifiers. That is bad since the more interactive and precisely timed communication we need, the more attackable, complicated, and delicate everything is, and the more that is required of each verifier – contrary to the desire that verifiers lead simple lives. Meanwhile, the BB-homo scheme requires *almost no*[24] communication and almost none of it is interactive – the homomorphic-totaling step can be carried out by only one party and is an easy computation (and other parties and verifiers could also carry it out) and the only cooperative part of the BB-homo scheme operates on only a very small amount of data (the election-total), and the verifiers can get away with doing only a small amount of work and communication.
3. **Study so far:** The BB-homo scheme has been more heavily studied both theoretically and experimentally than our JCJ-based voting scheme.

Applicability: The crypto-voting community unfortunately has consisted of computer scientists disconnected from

[23] An amount of order $V + N$.

[24] BB-homo employs only $O(1)$ communication, *aside* from (1) the initial communication between the voter and BB when each voter is voting (which is verified by the voter and BB themselves and results in a publicly posted ballot with ZK-validity proof) which is $O(V + N)$ in all, and (2) anybody making downloads from the BB. Although these communications also are of order $V + N$ items, the same count as our JCJ scheme, they are non-interactive. Also, verifiers can get away with only downloading and checking, say, 1000 random ballots from the BB, rather than the entire BB worth – that is still good enough to get reasonable confidence less than 0.1% of the ballots are fraudulent. So in a very real sense the communication requirements for BB-homo are much smaller than for our JCJ-based scheme.

political scientists. Therefore they have failed to recognize the existence of many other (superior) kinds of election methods besides the crudest method "plurality voting" (where a vote is the name of one candidate and the most-named candidate wins).[25] In particular, my own "reweighted range voting" [27] is an excellent example to keep in mind. RRV combines votes *non*additively[26] to determine winners, and RRV votes in an N-candidate election are N-tuples of real numbers. The latter fact allows voters to uniquify their votes by altering the low-significance bits of those real numbers. These properties of RRV are enough to break every available cryptographically secure voting scheme (except perhaps for ones completely infeasible in practice) including the one proposed here. More precisely, all schemes that employ mixnets to anonymize votes, are useable only with election systems in which the number of possible votes is much smaller than the number of voters.[27] Our new scheme does have the advantage that it is applicable to non-additive election systems such as "instant runoff voting," whereas schemes based on homomorphic encryption only work for additive elections.[28]

10 Resource estimates I – this is not practically feasible

Using the "mix and match" protocol for performing secure general multiparty circuit-computations [17], a G-gate boolean forward-flow logic circuit may be run on secret data (with ZK verification proofs produced, if desired[29]) by Q cooperating but mutually distrustful parties, in $O(QG)$ exponentiations in an elliptic curve group.

Our runtime is dominated by the cost of simulating a circuit for ElGamal decryption and AES encryption. Assume such a circuit contains $G = 10^{10}$ logic gates. Assume $O(Q) = 100$. Assume the number of votes cast is $V = 10^9$. The total computation required to run an election then is equivalent to the cost of performing about 10^{21} exponentiations in an elliptic curve group.

This is an enormous cost. Given that we plan to pay that cost, it surely will be worth designing and building special purpose hardware for performing exponentiations in an elliptic curve group. At present, software by Dan Bernstein can accomplish an exponentiation in the NIST-224 elliptic curve group in somewhere between 522000 and 1357000 Intel pentium cycles, depending on the processor and certain auxiliary conditions, which with a 4 GHz processor would take between 1/8 and 1/3 milliseconds. Suppose hypothetical special hardware can speed that up to 1 microsecond. Suppose further that we employ 10^7 such hardware devices in parallel. Then the total computation would take 10^{14} microseconds, i.e. 10^8 seconds, i.e. 3 years. Conceivably I have been too conservative in my estimates, in which case conceivably 100-times greater speed is achievable. That would reduce the time to "only" 12 days.

Even more horrible would be the prospect of *communicating* ZK-proofs of the validity of all those computations to verifiers who each have much less than government-scale resources. However, our verifiers could decide simply to *trust* the secure multiparty computations (based on the theory that enough of those parties are honest, or enough of them are distrustful enough of the others). That would greatly decrease the verification work and the communication requirements to the verifiers. However, the communication requirements among the mutually-distrustful talliers could *not* be eliminated, and they are huge. And since the talliers *are* mutually distrustful, they would need to be in secure, physically well-separated locations. The total number of bits communicated over long lines would be on the order of 10^{24}, which, even assuming 100-gigahertz bit rates, would take 3000 centuries.

Indeed, the original-flavor JCJ scheme, which employed $O(V^2)$ work, might have required say $100V^2$ exponentiations in an elliptic curve group, which with $V = 10^9$ would be 10^{20}. Thus the quadratic-work algorithm actually would consume comparable or less work than our linear-time algorithm if $V = 10^9$, because the original quadratic scheme had a far smaller constant hidden in its O thanks to the fact it does not employ SGMPC. (The linear-work method only starts to win once $V > 10^{10}$, which exceeds current world population.)

We conclude from this that despite the enormous *theoretical* improvement we have made by reducing JCJ's work requirement from quadratic to linear, and despite the fact that our scheme only uses $O(V + N)$ secure general multiparty computations that each operate on independent chunks of data of small bounded size (< 1Kbit), our scheme so far still is *not* practically feasible for use in large elections! Secure multiparty computation simply is infeasible for large scale practical use!

[25] We are happy to report the following recent exception [14].

[26] Some other examples [22][18] of non-additive election methods are "Instant Runoff Voting" (the single-winner case of Hare "single transferable vote") and Woodall's "Descending Acquiescing Coalitions" method [29]. Since IRV votes are rank-orderings of the candidates there are $N!$ (or more – approximately $N!e$ – if ranking only a subset of the candidates is permissible; even more if equalities are permitted in rankings) possible votes, which if $N! \gg V$ is enough to permit uniquifying your vote. Woodall-DAC votes in full generality would be *partial quasi-orders* among N candidates, of which there are approximately $2^{N^2/4}$; in a 10-candidate election the number of possible DAC votes is [26][8] exactly 8,977,053,873,043 which would seem quite sufficient to allow you to uniquify your vote even with the entire world population voting.

[27] In an N-candidate election, if plurality voting is used the number of possible votes is N. If "approval voting" is used, the number is 2^N (or 3^N for "approval voting with blanks"). If "Borda," "instant runoff voting," or other ranked-order systems are used, the number is $N!$. In "range voting" and related systems the number of possible votes is ∞^N, and even $(\infty + 1)^N$ if "blanks" are permitted.

[28] In principle, our new schemes is applicable to a *strict superset* of the election methods BB-homo schemes can handle. That is because any additive method can be "bit split" into a weighted sum of a constant number of independent binary yes-no elections, and those can be handled with the method of the present paper. In contrast, BB-homo schemes simply cannot handle non-additive election methods like IRV and Bucklin voting.

[29] This is a simple matter, since in all of these schemes each party must produce ZK proofs for the purpose of convincing the other distrustful parties that he is behaving correctly. So all we need to do is to broadcast these proofs to all verifiers in addition to just to the concerned parties.

11 Third idea: speedup to the point of practicality

The above cost analysis makes it clear that, if we want to turn this into something of practical, as opposed to merely of theoretical, interest, then we need to eliminate almost all SGMPC and replace it with far faster special purpose algorithms. Having $O(V+N)$ SGMPCs is too much. We need to reduce that count down to $O(1)$. Fortunately, it is possible to do that.

The key insight is that there is, essentially, only *one* kind of secure multiparty computation that we do a lot.[30] It consists of cooperatively ElGamal decrypting some shared-secret randomized-ElGamal-encrypted data (but not unsharing the resulting plaintext, so it remains secret) using an immaculately-shared-secret decryption key, and, then cooperatively re-encrypting (or merely secure-hashing [31]) that plaintext result in some deterministic but secret manner (i.e. using an immaculately-shared-secret encryption key), and then posting the final result to a BB.

Recall that an ElGamal encryption of a secret message M is the tuple (g^r, Mh^r) where g and h are the public keys, (g, h, and M are all elements of some public elliptic curve group of large public-prime order) and r is a random integer.

Re-encryptions can be performed by re-using the same g and h to get

$$(g^{r_1} g^{r_2} \cdots g^{r_m},\ h^{r_1} h^{r_2} \cdots h^{r_m} M) \tag{2}$$

which is just $(g^r, h^r M)$ where $r = r_1 + r_2 + \cdots r_m$. This fact allows our mixers to re-encrypt each message M that passes through them using new randomness r_j for each pass and each message, and without it being necessary for any mixer to know the plaintext M or to know the secret decryption key ℓ where $g^\ell = h$.

The decryption algorithm, only performable by somebody who knows ℓ where $h = g^\ell$, is to compute $(g^r)^\ell = h^r$ from the first tuple entry, then divide it out of the second tuple entry to compute M.

If, however, we do not wish to produce M, but merely a deterministically encrypted (or, better, hashed) version of it, then we could agree instead to do this:

1. Compute $(g^r)^{\ell z} = h^{rz}$ from the first tuple entry;
2. Compute $(h^r M)^z = h^{rz} M^z$ from the second tuple entry;
3. Divide to get M^z;
4. Output only a subset of M^z's bits to get a deterministic hash function of M.

In this procedure, M itself never is computed, but a hash (namely, the first 50% of the bits of M^z) of M is. This hash will now substitute for the roles played by F_K and F_J (and z will play the role of K or J) in the original protocol.[32] Here z, ℓ, and ℓz could be immaculately-shared secrets never known to anybody individually. (Since their generation and pre-sharing, and subsequent computation and broadcast of g and h, all could be done at $O(1)$ cost – which we regard as negligible – we ignore that issue.[33]) That way M would remain secret and unknowable.

Key point: It is still possible to raise group elements to the power z (or ℓz) cooperatively, for example (in the case of z) by each party successively raising it to a private power and providing a ZK proof they did so, where the product of these powers (not known to anybody) happens to be z.

Another way would be for each party to raise the original element to a private power where now the *sum* of those powers happens to be z. By making each party perform several such exponentiations and having various subset-sums among the private powers be z, we can make this work even if some parties cheat or refuse to cooperate [9][23][10][11] – then all results finally are multiplied at the end. Most generally, we could use a distributed "threshold ElGamal decryption" scheme (§A.6).

12 Resource estimates II – this now is practically feasible

This tremendously reduces the constant in our $O(V+N)$ work bound by eliminating SGMPC: The voting scheme now only takes about $100(V+N)$ exponentiations in an elliptic curve group. With $V = 10^9$ this is 10^{11} such exponentiations, which with the postulated 1μsec exponentiation hardware could be accomplished in 1.1 days even with no parallelism, and in 17 minutes with 100-fold parallelism.

The communication needs would be about $100(V+N)$ packets, each, say, 1 Kbit long, for a total of 10^{14} bits transmitted. This would take 10^5 seconds, i.e. 30 hours, if everything were transmitted over a single 1 GHz line.

Even with no special hardware, assuming 0.35 milliseconds per exponentiation all the work would take about 10 hours with 1000-fold parallelism. Buying 1000 computers at \$1000 per computer would cost \$$10^6$ which, assuming 10^8 voters, would cost 1 cent per voter.

This seems practical.

13 Acknowledgement

Roberto S. Araujo's questions about JCJ stimulated me to develop this idea. It was Araujo's suggestion that JCJ might be susceptible to a speedup, and he wrote most of §A.7.

An earlier draft of this paperand was based on a flawed approach to speed up JCJ. Once I understood how to construct

[30] Namely, this, or some subset of this, is done in our protocol steps 9, 15, 16, 19.

[31] Hashing seems a slightly superior idea to encrypting for our purposes since it should yield extra security since the plaintext is not reconstructible from a hash for *both* information-theoretic *and* computational-complexity reasons.

[32] In the special case $z = 1$ the above 4-step protocol becomes simply decryption, which is what is wanted in step 19 of the original protocol.

[33] Of course, were this paper's protocol to be implemented, one would want to devise good protocols for this part also. However, we refrain here, for two reasons: (1) it keeps our paper simple and general SGMPC theorems assure us that such protocols must exist. Even if they are inefficient this does not matter much because they are only run $O(1)$ times. (2) Section 3.2.4.2 of [15] and section 3.3 of [16] both already presented multiparty protocols to allow secret-sharers to create a random immaculate shared-secret triple (a, b, c) with $c = ab$ and a and b both uniform random (all mod P), and to ZK-prove that $c = ab$. Good protocols of this sort were first invented by Donald Beaver.

an attack on that earlier approach, the present paper's approach suggested itself as a simpler and flaw-free version of the earlier one.

References

[1] Masayuki Abe: Mix-networks on permutation networks, ASIACRYPT (1999) 258-273, Springer LNCS #1716. Corrections and extensions: Masayuki Abe & Fumitaka Hoshino: Remarks on Mix-Network Based on Permutation Networks, Public Key Cryptography (2001) 317-324, Springer (LNCS #1992).

[2] M.Bellare & S.Micali: Non-interactive oblivious transfer, pp. 547-557 in CRYPTO 89=Springer LNCS #435.

[3] 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.

[4] 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`.

[5] Fabrice Boudot: Efficient Proof that a Committed Number Lies in an Interval, pp.431-444 in Proc. of EuroCrypt 2000, Springer Verlag LNCS #1807.

[6] 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.

[7] 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.

[8] S.K.Das: A Machine Representation of Finite T_0 Topologies, J.ACM 24,4 (1977) 676-692.

[9] Yvo G. Desmedt & Yair Frankel: Threshold cryptosystems, Crypto 89 (1990) 307-315 (Springer LNCS #435).

[10] Yvo G. Desmedt: Threshold cryptography, European Trans. Telecommunications 5,4 (1994) 449-457.

[11] Yvo G. Desmedt: Some recent research aspects of threshold cryptography, Information Security Proceedings (1997; Springer LNCS #1396) 158-173.

[12] 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.

[13] Jens Groth: A Verifiable Secret Shuffle of Homomorphic Encryptions, pp. 145-160 in *Practice and Theory in Public Key Cryptography - PKC 2003* (Springer LNCS #2567).

[14] Jens Groth: Non-interactive Zero-Knowledge Arguments for Voting, pp. 467-482 in *Applied Cryptography and Network Security - ACNS 2005* (Springer LNCS #3531).

[15] 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.

[16] 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/`.

[17] Markus Jakobsson & Ari Juels: Mix and match: secure function evaluation via ciphertexts, Asiacrypt (2000) 162-177, Springer (LNCS #1976).

[18] J. Economic Perspectives 9,1 (1995) is a special issue on voting methods and is a recommended way to learn about them.

[19] Ari Juels, Dario Catalano, M.Jakobsson: Coercion-resistant electronic elections. One older version was at Cryptology ePrint Archive: Report 2002/165 `http://eprint.iacr.org/`; latest version on Juels web page: `http://www.rsasecurity.com/rsalabs/node.asp?id=2030` as of June 2005.

[20] Donald E. Knuth: Sorting and Searching (Art of Computer Programming vol. 3) Addison-Wesley 2nd ed. 2003.

[21] Ph.D. MacKenzie, Th.Shrimpton, M.Jakobsson: Threshold Password-Authenticated Key Exchange, CRYPTO (2002) 385-400.

[22] Hannu Nurmi: Voting procedures: A summary analysis. British Journal of Political Science 13,2 (1983) 181-208.

[23] T.P.Pedersen: A threshold cryptosystem without a trusted party, Eurocrypt 91 (Springer LNCS #547) 522-526.

[24] Claus-Peter Schnorr: Efficient Signature Generation by Smart Cards, J. Cryptology 4,3 (1991) 161-174.

[25] Adi Shamir: How to share a secret, Commun.ACM 22,11 (1979) 612-613.

[26] N. J. A. Sloane: The On-Line Encyclopedia of Integer Sequences (2005) `http://www.research.att.com/~njas/sequences/`. See sequence A000798.

[27] Warren D. Smith: Reweighted Range Voting – new multiwinner election scheme, #78 at `http://math.temple.edu/~wds/homepage/works.html`.

[28] Warren D. Smith: Cryptography meets voting, #80 at `http://math.temple.edu/~wds/homepage/works.html`.

[29] Douglas R. Woodall: Monotonicity of single seat preferential election rules, Discrete Applied Maths. 77,1 (1997) 81-98.

A Appendix: Highly abbreviated review of zero knowledge proofs and other known tools

In the main text we have alluded to (or assumed use of) various known cryptographic tools which we did not explain there. We now provide details about them. Some of our remarks will also be important for achieving good efficiency in practice. The survey [28] gave the details of many such ZK-proofs and cryptographic voting schemes, including BB-homo voting schemes, so most of the things we discuss in this appendix are also treated there, usually more extensively and with more sources cited.

A.1 ZK proofs of same exponent

Suppose Peter Prover knows that two publicly known quantities $x = g^\ell$ and $y = h^\ell$ have the *same* discrete logarithms ℓ (to publicly known respective bases g and h) in some group of large[34] prime order P. (We recommend working in an elliptic curve group.) 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:

1. Peter chooses random r mod P (but keeps it private);
2. Peter computes and prints $a = g^r$ and $b = h^r$;
3. Vera chooses random c mod P and tells it to Peter;
4. Peter computes and prints $z = r + \ell c$ mod P;
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. (But observe: if Peter knew Vera's c in advance, then he could have chosen r nonrandomly and "forged" a proof.)

This protocol can be made non-interactive by the Fiat-Shamir trick: make the challenge c instead be a standard secure hash function of (x, g, y, h, a, b), which *Peter* computes and publishes, and Vera merely verifies.

A.2 Applications to ZK proof of encryption, ZK proof of knowledge of plaintext

By using the above NIZK protocol, Peter can convince Vera that he has produced an ElGamal encryption $(g^\ell, h^\ell M)$ of a message M provided by Vera, but without revealing the secret key ℓ (note that the group elements g and h are public keys). Or he can show that $(g^{r+\ell}, h^{r+\ell}M)$ is an ElGamal *re*-encryption of the original encryption, without revealing r.

Also, he can prove knowledge of ℓ in the encryption $(g^\ell, h^\ell M)$, thus proving knowledge of the plaintext M, but again without revealing either ℓ or M.

A.3 ANDing and ORing ZK proofs

To ZK-prove the logical AND of two claims, simply present ZK proofs of them both. What we call an *indecomposable* AND of two NIZK proofs involves "challenges" inside it that are constructed from a secure hash function of *both* component proofs. The point of this is that some enemy cannot now surgically excise the component NIZK proofs and glue them together with other components to get his own NIZK ANDed proof of something else – well, he can, but the resulting proof will not have the hashing property and hence can be immediately revealed as having been produced by surgery by somebody unauthorized, as opposed to having been produced by an original authorized prover.

Logical ORing is trickier. Roughly speaking (adjustments may be needed in particular cases), the procedure is summarized by the mnemonic

$$\text{ZK-proof}_c(A \vee B) \equiv \text{ZK-proof}_d(A) \wedge \text{ZK-proof}_e(B) \wedge \{d + e = c\} \tag{3}$$

where the subscripts c, d, e of the proofs denote the integer "challenges" presented to the prover by the verifier. The original proofs considered independently each had their own challenges d and e. But the combination proof has only one challenge $c = d+e$. The reason this works: the prover can "forge," i.e. produce himself ahead of time, one of the challenges d OR e, and hence finds it feasible to "fake" that ZK-proof; but the other challenge is then determined by the linear[35] constraint $d + e = c$ and the verifier's randomness in choosing c, and hence is infeasible to forge. (The prover publishes the values of d and e.) That forces the prover to genuinely prove one of the component proofs. If the challenges in the component proofs in fact could be got non-interactively via a secure hash function (an example of this was above) then the combination ZK-proof also can be made non-interactive.

A.4 Designated-Verifier ZK-proofs

A brilliantly simple idea. To ZK-prove statement X in such a way that only Bob will believe your proof: ZK-prove: "X *OR* (proof of knowledge of Bob's secret key)."

Bob's thoughts: "of course, this person does not know my secret key, so X must be true."

Alice's thoughts: "Bob could have told this person his key (actually in typical use 'this person' *is* Bob). So I have no reason to believe X."

Note that Bob cannot re-use the proof he is given to convince anybody else of X.

A.5 Commitments

Somebody can "commit" n bits of information by publishing an AES-like encryption of a $(n + 2s)$-bit message consisting of those n bits padded with s one-bits followed by s random bits where s is a security parameter. He can later "reveal" the committed bits by revealing his (s-bit) secret encryption/decryption key. (Other protocols also are possible, e.g. [12].)

A.6 Cooperative decryption [9]

In threshold ElGamal decryptions, at least t (for some t with $1 \le t \le m$) decryptors need to cooperate to decrypt a message [9]. This can be accomplished by having the decryption exponent K be the constant term $P(0)$ of a degree-$(t-1)$ polynomial where decryptor j knows $K_j = P(j)$ 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 exponentiations to the secret K_j powers followed by public exponentiations to the L_j power and a final producting step to combine it all together:

$$x^K = x^{\sum_j L_j K_j} = \prod_j (x^{K_j})^{L_j}. \tag{4}$$

[34] "Large" means $\approx 2^{256}$. We recommend elliptic curve groups [4]. Warning: We use multiplicative group notation throughout, although most elliptic curve authors use additive group notation.

[35] Actually the "+" could instead be a bitwise XOR or a group-multiplication – any of these, in general, would yield a valid OR-combination proof.

Each decryptor j should broadcast NIZK-proofs he really is exponentiating with his correct private exponent K_j. Note: with this scheme one decryptor will be able to cheat *once* by using a bogus K_j (thus learning the product but not revealing it to the others) but then immediately will be spotted as a cheater and permanently excluded from the protocol. This cheat effort could perhaps break privacy on *one* vote, at tremendous personal cost to the cheater. This seems an acceptably small security leak. The small magnitude of this leak could be reduced further by forcing all decryptors to "commit" their x^{K_j} values before revealing them, then unveiling the commitments in "verifiably random" order. That way, a cheating decryptor would have a $\geq (t-1)/m$ chance of being unveiled as a cheater before enough other information was available for him to gain the benefit from that cheating. Finally, (albeit expensively) if we desire the leak-magnitude could be reduced further still, to essentially zero, by use of "bit-by-bit revelation" techniques (§A.9).

A.7 Plaintext equality test (PET):

An important primitive, used in [21][17] and needed by the voting protocol presented in [19] verifies that two ciphertexts (ElGamal encrypted using the same public key but different randomness) are encryptions of the same plaintext. This is achieved by dividing the two ElGamal encryptions and verifying that the results encrypt the value 1. Thus, let $(\alpha, \beta) = (g^r, M_1 h^r)$ and $(\gamma, \delta) = (g^s, M_2 h^s)$ be the two ElGamal ciphertexts where r and s are random; if $M_1 = M_2$, then $(\alpha/\gamma, \beta/\delta) = (g^{r-s}, 1h^{r-s})$.

To complete the verification, the resulting encryption $(g^{r-s}, 1h^{r-s})$ must be proved to encrypt the value 1. That can be accomplished by anybody who knows the decryption key ℓ where $h = g^\ell$ (so $M = Mh^r/(g^r)^\ell$); or by joint decryption by mutually distrustful parties who had previously secret-shared [25] the ElGamal decryption key. Since $1^z = 1$ whereas with high probability in a group of large prime order $x^z \neq 1$ if $x \neq 1$ and z is random, it suffices to produce, not the decryption itself, but rather a random power of it ($M^z = (Mh^r)^z/((g^r)^z)^\ell$), thus definitely revealing zero knowledge about the plaintext even if the "random" quantities r and s had in fact been maliciously chosen.

A ZK proof that the correct decryption was in fact produced, i.e. that the correct exponent ℓ was in fact used (but without revealing ℓ) could then be produced with the aid of the preceding "ZK proof of same exponent." (Similarly we ZK-prove that the same exponent z was used both times in the version with z's.)

A.8 ZK-proofs of ballot validity and interval membership

In a yes-no election, a valid vote is an encryption of "1" or "0." A voter could provide an ORed ZK-proof that some ElGamal cryptotext $(g^r, h^r k^M)$ encrypts either $M = 1$ or $M = 0$, but without revealing which.

In an election in which votes consist of integers in a range $[0, 2^b - 1]$, i.e. b-bit integers, the voter could simply provide the elementwise product of b ElGamal 2-tuples,

$$\prod_{j=0}^{b-1} (g^{r_j}, h^{r_j} k^{2^j M_j}) \tag{5}$$

where each M_j was proved as before to be a 1-bit number.

Boudot [5] discussed more general and supposedly more efficient (for large b) interval-membership ZK-proofs than this simple procedure. However, his "more efficient" procedure actually is "less efficient" and "more complicated" than this, because Boudot's methods depend for their security on the assumption that integer factorization is hard, whereas we depend on the assumption that discrete logarithms in elliptic curve groups are hard.[36] That allows us to use much shorter key lengths to get the same security, causing just *one step* in Boudot's method to take more work than our entire procedure – rendering irrelevant the fact that Boudot's methods have fewer steps! However, [28] both pointed out this error and repaired it by devising replacements for Boudot's ZK-proof components that depend instead on elliptic-curve discrete logarithm assumptions.

Efficient ZK-validity proofs for "approval voting" and "Borda voting" (and several other kinds of voting) type ballots are discussed in [14]; many of them are based on mixnets.

Fortunately, to a large extent we *do not need to worry* about possible "data format incompatibilities" between our election system and somebody's ZK ballot-validity proof, because our election system *does not care* how the ballots are encrypted, so long as they can be mixed and ultimately decrypted. Groth [14] worried extensively about that issue because he was interested in making his ballots compatible with later processing by homomorphic-encryption vote-totaling schemes [6][7]. We do not have to. Therefore, we can handle *any* voting method with a finite number of possible types of votes by simply encoding the vote as an integer (if there are 55 possible votes, then an integer in $[0, 54]$) in which case an interval-containment ZK-proof is all we need.

Similarly, we also do not particularly care what underlying vote-combining system (plurality, Borda, approval, etc. [22][18]) is employed. That is an advantage of systems like ours that ultimately reveal and post all the plaintext votes on a BB. All of Groth's ballot-validity proofs [14] are useable.

A.9 Bit-by-bit revelation

Suppose several parties have their own individual secrets, and wish to reveal them to each other. However, each trembles in horror of the possibility that he will foolishly reveal his own secret *before* the others, allowing them (while laughing hysterically at his idiocy) to suddenly terminate the protocol and learn his secret.

This problem is soluble by having each party release one bit of his secret at a time (along with a ZK-proof the bit was valid), thus causing no party to be able to quit with more than a single-bit advantage. If each secret is a b-bit integer known to all in encrypted form as in EQ 5, then one can simply reveal another bit M_j each time. Recall each M_j had already been ZK-proven to *be* a single bit via an OR of ZK-proofs it was

[36] Please regard all the multiplicands in EQ 5 as elements in an elliptic curve group with multiplicative group-notation.

either 1 or 0. By revealing the two component proofs (one of which was "forged") in full rather than just their OR, that M_j becomes revealed to all.

A.10 Shamir's Secret sharing [25]

The dealer who wants to share a secret S selects a random polynomial

$$F(x) = S + r_1 x + r_2 x^2 + \cdots + r_{T-1} x^{T-1} \tag{6}$$

of degree $< T$, and sharer j gets[37] $S_j = F(j)$ as his share of the secret for $j = 1, 2, \ldots, Q$. Here the r_k are random integers mod P for some public prime P, and S is another. Any T sharers can reconstruct $F(x)$ and hence determine S by polynomial interpolation mod P, but $T-1$ sharers are insufficient. This scheme is "linear," i.e. a summed secret $Q + S$ corresponds to summed secret shares $Q_j + S_j$. Immaculate shared secrets S can be got by having each sharer generate his own random secret, then (acting as dealer) deal it out, and then the sum of all of them is S.

The scheme as we have described it so far is vulnerable to cheating dealers (who distribute bogus shares and thus do not really reveal their secret) or cheating sharers (who "reveal" bogus shares and thus learn the secret while honest players do not). "Verifiable" secret-sharing schemes [12][15] do not have those weaknesses. They require the dealer to commit his secret before dealing it out, and commit to all the shares he deals out, and to ZK-prove that the share-commitments legitimately correspond to the committed secret. They also require the sharers who decide to reveal their shares, then to open the share commitments, thus proving they are not revealing a bogus share.

A.11 Mixers

A "mixer" inputs N encrypted data items and outputs those same N items, only in a shuffled order and with each data item re-encrypted. The mixer wishes to convince everybody (by providing a ZK-proof) that he just did exactly that, but without revealing the shuffling permutation or the re-encryption transformations.

Unfortunately JCJ [19] cited Furukawa-Sako which (its authors later realized) is a flawed mixer scheme because their proofs actually are not zero-knowledge. They also cited a complicated mixnet scheme due to Neff, which I do not have confidence in. The allegedly best mixer scheme now available is by Groth [13] – and it seems perfectly suited to our purposes since it employs ElGamal encrypted inputs and (shuffled re-encrypted) outputs with the same ElGamal public keys in both cases.

Unfortunately Groth's mix-scheme is also complicated and I do not understand it fully.[38]

Therefore I remark that three *extremely simple* mixnet schemes, all of which can employ *non-interactive*[39] validity ZK-proofs, are described in [28] sections 4.12 (first two schemes) and 7.1 (third scheme, similar to Abe's [1]). Their simplicity is achieved at the respective *costs* of either

① A large constant hidden in the "$O(n)$" work bound.
② Allowing the mixer to have a reasonable probability of "cheating" on a bounded number of votes (m added, deleted, or altered "cheat votes" $\Longrightarrow$ probability $\geq 1 - 2^{-m}$ of getting caught), and furthermore this scheme reveals N bits worth of partial knowledge about the shuffle, or
③ causing the total work to be of order $N \log N$ (and with order $\log N$ mixer stages) rather than the optimal $O(N)$ and $O(1)$ achieved by [13].

I believe that if N is large, then in practice for our voting needs, these imperfections do not matter and are worth accepting to gain the benefits of simplicity.

We now explain scheme ②, because it seems to be the most practically useful scheme available for our voting purposes.

1. Shuffler outputs all N elements of B, and C ($2N$ in all) in encrypted form. Here A is the N encrypted input items, B is the N re-encrypted output items, and C is a third random shuffling and re-encryption of those N items.
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 re-encryptions used to generate the red C's from the corresponding A's (and reveals the correspondences) and similarly to generate the B's that come from the blue C's.

After this, the verifier is confident – unless the shuffler can break the cryptosystem or enjoyed an exceedingly vast amount of luck – 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 exceptional unshuffled or wrongly-encrypted elements.

This scheme can be made completely *non-interactive* by having the shuffler generate κ himself as a standard cryptographic hash function of everything he output in step 1; and provided N is larger than the security parameter (encryption-key bitlength) the scheme remains secure. However, because an evil shuffler can usually make a probability-ϵ event happen by "working $1/\epsilon$ times harder" retrying different shuffle+proof attempts, for our voting purposes it seems preferable in practice to use the *almost* non-interactive protocol we just gave. The challenge κ in practice would best be produced by a consortium of verifiers by applying a standard secure hash function to everything the shuffler output in stage 1, *padded* with random bitstrings then contributed by each verifier.

The scheme we just gave has the disadvantage that it *reveals* $\approx N$ *bits of partial knowledge* about the shuffle, namely we

[37] Via transmission over a private channel or publicly via cryptography.

[38] We also note that Groth claimed his scheme involved $\approx 6N$ exponentiations in the proof and another $6N$ in the verify, for $12N$ total, supposedly a new record improving on the previous best total $18N$ by Furukawa & Sako. However, Groth neglected to mention that F&S had pointed out that their $18N$ was reducible "'to the work-equivalent of $\approx 5N$ exponentiations," and also neglected to mention that F&S's scheme was flawed. Groth also uncritically cited Neff.

[39] Groth claimed in email to WDS that his mix can be made non-interactive using a Fiat-Shamir-type trick. However, this issue was not addressed in his paper [13].

know that the $\lfloor N/2 \rfloor$ red items do *not* get shuffled to the $\lceil N/2 \rceil$ blue locations. This "disadvantage" hardly seems to matter for us (since, e.g, N bits is asymptotically negligible compared to the number $\log_2 N!$ of bits required to specify the shuffle fully). But if you really care, it could be eliminated (at the cost of $O(N)$ additional proof and verification work) by ZK-proving, for each of the red C-items, that it corresponds to either of *two* specified A-items, but without saying which (and let all these A-pairs be disjoint). That can be done using an "OR of two ZK-proofs."[40]

Another (tiny) disadvantage of this scheme (which again should not matter in our voting application[41]) 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 desired integer $f \geq 1$. Alternatively, the cheat-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 C-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 of those proofs he actually wanted to examine (which again could be done using a very short challenge used as a seed for a standard strong pseudorandom number generator), and then in stage 5 the prover would produce only the requested proofs. This "partial verification" still seems adequate in most large scale voting applications even when $f = 0.02$ and is tremendously less costly – in the $f \to 0+$ limit the shuffler's total work reduces to only the $4N$ exponentiations needed to perform the $2N$ ElGamal re-encryptions needed to produce B and C; meanwhile the verifier's work is of order $O(Nf)$. The shuffler→verifier communication requirements are $O(Nf)$ group-elements and the verifier→shuffler communication requirements are equivalent to $O(1)$ group-elements.

When mixing *tuples* we need to make the mixer provide validity ZK proofs that it is indeed permuting tuples bodily; this can be done trivially by ANDing the usual ZK-proofs for tuple components. Also, we remark that Groth's mix-scheme (see the final two sentences of [13]) can be efficiently *re-used* for the purpose of proving additional batches of N data were mixed using the *same* shuffle-permutation.

A.12 Secure general-purpose multiparty computation (SGMPC)

The goal of SGMPC is to set up a protocol under which several mutually distrustful parties can perform *any* computation (specified by an *arbitrary* forward-flow circuit made of boolean logic gates) in such a way that ① the input bits, output bits, and intermediate bits all only are available in encrypted form throughout the entire process, so that no individual finds it feasible to deduce the unencrypted form of any of those bits. Further, ② this should be done in such a way that each party, and indeed any outside observer, is convinced that the computation was carried out correctly, and ③ a super-threshold subset of the parties can (if they agree to do so) decrypt any particular bit.

The first demonstration [3] that SGMPC is possible was based on Shamir secret-sharing [25] of each bit, with SGMPC protocols devised to perform 2-bit logic operations on secret-shared bits. In fact, they even showed how to add, multiply, and generate immaculate random shared-secret finite-field-elements. The addition and randomization methods are trivial, but the multiplication method is complicated (see section 3.1 of [15] for an excellently clear description).

Donald Beaver invented a brilliantly simple idea for avoiding the difficulty of multiplying shared secrets. It involves a preparatory phase of first generating and distributing a pool of triples (a, b, c) of random constants satisfying $c = ab$ to all secret-sharers. Beaver's trick made SGMPC far closer to being practical [16].

Joe Kilian gave another demonstration of SGMPC in which everything was based on only *one* remarkably simple-sounding primitive cryptographic operation, which Kilian called "oblivious transfer." Peter→Vera OT 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 looks inside it; the other envelope is burned without opening it.

Bellare and Micali [2] showed how to implement OT using an ElGamal-crypto-like software protocol. OT also can be implemented using trusted physical devices, e.g. actual envelopes, quantum cryptography, etc. However, Kilian's OT-based construction, while perhaps important theoretically, appears too inefficient and complicated to be a practically important way to do SGMPC.

The presently best known (in terms of efficiency and simplicity) method of implementing SGMPC is the "mix and match" method [17]. The full technique is too complicated to describe here, so we only sketch it. It involves ① producing (by means of a mixnet) an equivalent logic circuit but with "randomly scrambled truth tables" for the logic gates; this circuit is not known to any individual party because the truth tables are stored in encrypted form; ② we *match* the input-bit of each logic gate with the output-bit of its predecessor gate, (or if there is no predecessor-gate, with the initial encrypted input bits to the whole circuit) by means of distributed plaintext-equality-tests; ③ Finally, the last gate produces the output bit in encrypted form; the players jointly decrypt it (and any other bits they want to publish in un-encrypted form). The joint decryptions need to be accompanied by ZK-proofs by each player that they are correctly doing their part in each.

The whole mix and match protocol requires $O(QG)$ modular exponentiations worth of work to produce a verification of circuit operation (for a circuit with G gates, with Q distrustful computing parties), assuming nobody cheats. If anyone cheats, the cheater is immediately spotted and the procedure is terminated. (It can be restarted with the corrupt party excluded.)

The work-expansion factor involved in converting a circuit

[40] 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.

[41] Because the benefit to the cheating shuffler of being able to mess with a few votes, is not worth the tremendous cost to him of being revealed as a proven fraud, if the probability of the latter is at all substantial.

from normal to SGMPC operation, although "bounded," is very large – each bit operation becomes $O(Q)$ modular exponentiations, i.e. about $O(Q) \cdot 10^9$ bit operations for a 10^9-gate circuit! For this reason, it presently only is technically/economically feasible to run very *small* computations in SGMPC mode.