Efficient Protocols based on Probabilistic Encryption using Composite Degree Residue Classes

Ivan Damgård and Mads Jurik

Aarhus University, BRICS*
(preliminary version)

Abstract. We study various applications and variants of Paillier's probabilistic encryption scheme. First, we propose a threshold variant of the scheme, and also zero-knowledge protocols for proving that a given ciphertext encodes a given plaintext, and for verifying multiplication of encrypted values.
We then show how these building blocks can be used for applying the scheme to efficient electronic voting. This reduces dramatically the work needed to compute the final result of an election, compared to the previously best known schemes. We show how the basic scheme for a yes/no vote can be easily adapted to casting a vote for up to t out of L candidates. The same basic building blocks can also be adapted to provide receipt-free elections, under appropriate physical assumptions. The scheme for 1 out of L elections can be optimised such that for a certain range of parameter values, a ballot has size only $O(\log L)$ bits.
Finally, we propose a variant of the encryption scheme, that allows reducing the expansion factor of Paillier's scheme from 2 to almost 1.

1 Introduction

In [7], Paillier proposes a new probabilistic encryption scheme based on computations in the group $Z^*_{n^2}$, where n is an RSA modulus. This scheme has some very attractive properties, in that it is homomorphic, allows encryption of many bits in one operation with a constant expansion factor, and allows efficient decryption. In this paper we propose some tools for applications of this scheme and some variants making it suitable for additional applications.

We propose a threshold variant of Pallier's scheme, allowing a number of servers to share knowledge of the secret key, such that any large enough subset of them can decrypt a ciphertext, while smaller subsets have no useful information. It is straightforward to apply known techniques for threshold RSA (such as [8]) to Pallier's scheme as it was originally described. However, doing this would lead to an insecure scheme as we explain later. We present a modification of Pallier's decryption algorithm that solves this problem. In work independent from, but

* Basic Research in Computer Science, Center of the Danish National Research Foundation

earlier than ours, Fouque, Poupard and Stern [4] propose the first threshold variant of Pallier's scheme. Their basic idea is very similar to ours, with only minor technical differences making our decryption algorithm slightly simpler (as detailed later).

We also propose a zero-knowledge proof of knowledge allowing a prover to show that a given ciphertext encodes a given plaintext. From this we derive other tools, such as a protocol showing that a ciphertext encodes one out of a number of given plaintexts. Finally, we propose a protocol that allows verification of multiplicative relations among encrypted values without revealing extra information.

We look at applications of this to electronic voting schemes. A large number of such schemes is known, but the most efficient one, at least in terms of the work needed from voters, is by Cramer, Gennaro and Schoenmakers [3]. This protocol provides in fact a general framework that allows usage of any probabilistic encryption scheme for encryption of votes, if the encryption scheme has a set of "nice" properties, in particular it must be homomorphic. The basic idea of this is straightforward: each voter broadcasts an encryption of his vote (by sending it to a bulletin board) together with a proof that the vote is valid. All the valid votes are then combined to produce an encryption of the result, using the homomorphic property of the encryption scheme. Finally, a set of trustees (who share the secret key of the scheme in a threshold fashion) can decrypt and publish the result.

Paillier pointed out already in [7] that since his encryption scheme is homomorphic, it may be applicable to electronic voting. In order to apply it in the framework of [3], however, some important building blocks are missing: one needs an efficient proof of validity of a vote, and also an efficient threshold variant of the scheme, so that the result can be decrypted without allowing a single entity the possibility of learning how single voters voted[1].

These building blocks are precisely what we provide here. Thus we immediately get a voting protocol. In this protocol, the work needed from the voters is of the same order as in the original version of [3]. However, the work needed to produce the result is reduced dramatically, as we now explain. With the El Gamal encryption used in [3], the decryption process after a yes/no election produces $g^R \bmod p$, where p is prime, g is a generator and R is the desired result. Thus one needs to solve a discrete log problem in order to find the result. Since R is bounded by the number of voters M, this is feasible for moderate size M's. But it requires $\Omega(\sqrt{M})$ exponentiations, and may certainly be something one wants to avoid for large scale elections. The problem becomes worse, if we consider an election where we choose between L candidates, $L \geq 2$. The method given for this in [3] is exponential in L in that it requires time $\Omega(\sqrt{M}^{L-1})$, and so is prohibitively expensive for elections with large L.

In the scheme we propose below, this work can be removed completely. Our decryption process produces the desired result directly. We also give ways to

[1] In [4], voting was also pointed out as a potential application, but no suggestion was made there for a proof of validity of a vote.

implement efficiently constraints on voting that occur in real elections, such as allowing to vote for precisely t out of the L candidates, or to vote for up to t of them. The complexity of this scheme is linear in L. We propose a particularly efficient variant of 1 out of L voting which is some cases will produce votes of length $O(\log L)$ bits. This is optimal up to a constant factor, since with less than $\log L$ bits one cannot distinguish between the L candidates. Furthermore this scheme requires only 1 decryption operation, even when $L > 2$.

In [6], Hirt and Sako propose a general method for building receipt-free election schemes, i.e. protocols where vote-buying or -coercing is not possible because voters cannot prove to others how they voted. Their method can be applied to make a receipt-free version of the scheme from [3]. It can also be applied to our scheme, with the same efficiency gain as in the non-receipt free case.

Finally, we propose a variant of the encryption scheme, that allows reducing the expansion factor of Paillier's scheme from 2 to almost 1.

2 Paillier's Probabilistic Encryption Scheme

For completeness, we recall here briefly Paillier's scheme, please refer to [7] for proofs of the claims we make here.

Paillier's scheme starts from the observation that if $n = pq$, p, q odd primes, then $Z^*_{n^2}$ as a multiplicative group is a direct product $G \times H$, where G is cyclic of order n and H is isomorphic to Z^*_n. Thus, the factor group $\bar{G} = Z^*_{n^2}/H$ is also cyclic of order n. For an arbitrary element $a \in Z^*_{n^2}$, we let $\bar{a} = aH$ denote the element represented by a in the factor group $\bar{G}$.

It turns out that the element $\overline{1+n} := (1+n)H \in \bar{G}$ is a generator, and moreover $(1+n)^i = 1 + in \bmod n^2$. Hence the cosets of H in $Z^*_{n^2}$ are

$$H, (1+n)H, (1+2n)H, ..., (1+(n-1)n)H.$$

This gives a natural numbering of these cosets.

Let λ be the least common multiple of $p-1$ and $q-1$, thus raising to exponent λ in Z^*_n always gives 1. Then if for some element a is in the i'th coset, or in other words $\bar{a} = \overline{1+n}^i$, then for some element in $h \in H$ it holds that $a = (1+n)^i h \bmod n^2$, and so

$$a^\lambda = ((1+n)^i h)^\lambda = 1 + (i\lambda \bmod n) \cdot n \bmod n^2$$

If we define the function $L()$ as $L(b) = (b-1)/n$, we get that

$$L(a^\lambda) = \lambda i \bmod n$$

So knowledge of the factors of n, and hence of λ, allows to compute discrete logarithms to the base $\overline{1+n}$ in $\bar{G}$. Clearly, if we have another generator $\bar{g}$ of $\bar{G}$, where $\bar{g} = \overline{1+n}^j$ then for arbitrary a, we have for some $x \in Z_n$

$$a = \bar{g}^x = \overline{1+n}^{jx}$$

and so discrete logarithms to base $\bar{g}$ can be computed by computing first the discrete log base $\overline{1+n}$, and then multiplying by $j^{-1} \bmod n$.

Paillier's public key system now uses n, g as public key, where g is a random number modulo n^2, chosen such that $\bar{g}$ generates $\bar{G}$. The secret key is λ. To encrypt a number $i \in Z_n$, one chooses a random $r \in Z^*_{n^2}$, and sends

$$E(i,r) = g^i r^n \bmod n^2$$

From the structure of the group we are in, it is clear that $r^n \bmod n^2$ is a random element in H, and so $E(i,r)$ is a random element with the property that $\overline{E(i,r)}$ has discrete log i to the base $\bar{g}$ in $\bar{G}$.

By what we said above, it is now clear that the receiver who knows λ can compute i as

$$i = L(E(i,r)^\lambda \bmod n^2) \cdot L(g^\lambda \bmod n^2)^{-1} \bmod n$$

This system relies for security on the difficulty of distinguishing random elements in different cosets of H, it is semantically secure, if it is hard to decide if two given elements are in the same coset. This problem is at most as hard as factoring n, but we do not know much else about it. We hope that our results will motivate further study of it.

3 Some Building Blocks

3.1 A Threshold Variant of the Scheme

What we are after in this section is a way to distribute the secret key to m servers, such that any subset of at least t of them can do decryption efficiently, while less than t have no useful information. Of course this must be done without degrading the security of the system.

In [8], Shoup proposes an efficient threshold variant of RSA signatures. The main part of this is a protocol that allows a set of servers to collectively and efficiently raise an input number to a secret exponent modulo an RSA modulus n. A little more precisely: on input a number a, each server returns a share of the result, together with a proof of correctness. Given sufficiently many correct shares, these can be efficiently combined to compute $a^d \bmod n$, where d is the secret exponent.

As we explain below it is quite simple to transplant this method to our case, thus allowing the servers to raise to a secret exponent modulo n^2. It now may seem easy to solve our problem: We first let the servers help us compute $E(i,r)^\lambda \bmod n^2$ and $g^\lambda \bmod n^2$, then the remaining part of the decryption is easy to do without knowledge of λ.

However, the natural goal for a threshold version of an encryption scheme is, informally speaking, to be as secure as the original scheme. Unfortunately the simple idea we just described is completely insecure under a chosen ciphertext attack. In such an attack, $E(i,r)^\lambda \bmod n^2$ will become known to the adversary.

Now, if he submits as ciphertext a number of form $(1+n)^i r^n \bmod n^2$, then he will learn $\lambda i \bmod n$, he can divide out i to learn λ, and the system is completely broken. Applying the same attack to Pallier's original system does not lead to a similar disaster: it is easy to see that attacker would learn the discrete log of $\bar{g}$ base $\overline{(1+n)}$, but it is not at all clear that this would enable him to find the secret key. If we restrict to attacks where the adversary knows the plaintext for all the ciphertexts he submits, then the attacker learns nothing new when the original scheme is used, but learns the discrete log of $\bar{g}$ base $\overline{(1+n)}$ in case of the naive threshold variant.

So clearly, this simple scheme is not as secure as the original one, and we need to be more careful. The natural solution is to conduct the entire decryption algorithm in a multiparty fashion, such that the only piece of information that the servers release is the plaintext. However, the decryption algorithm consists, first of computation modulo n^2, then some integer arithmetic, and finally computation modulo n. Doing this in a distributed way seems to require the full power of general multiparty computation, and would certainly require heavy interaction between the servers, something we definitely want to avoid in a practical application. We therefore propose a way to change the decryption algorithm such that one exponentiation essentially produces the plaintext directly without releasing extra information.

When the keys are generated, we first choose n, g as before. For consistency with the following, let us assume that $n = pq$, where $p = q = 3 \bmod 4$. This means that $\lambda = 2m$ where m is odd. Then we compute j such that $\bar{g} = \overline{1+n}^j$ using our knowledge of λ. Finally, we compute a number d such that $d = 0 \bmod m$ and $d = j^{-1} \bmod n$. This is easy by the Chinese Remainder theorem, since n and m are relatively prime. Note also that since $\bar{g}$ generates $\bar{G}$, j must be prime to n. We can now see that raising to exponent $2d$ essentially produces the decryption directly. By choice of d and the fact that g can be written as $g = (1+n)^j h$ for some $h \in H$, we get:

$$E(i,r)^{2d} = (1+n)^{2jid}(h^i r^n)^{2d} = (1+n)^{2jid \bmod n} = (1+n)^{2i} = 1 + 2in \bmod n^2$$

and so $L(E(i,r)^d \bmod n^2)/2 = i \bmod n$. The last computation of $L()$ is trivial and easy to reverse, so now the result of the exponentiation does not reveal any extra information.

We now briefly explain how the method from [8] can be used in our context: In [8] a method for distributed RSA signatures is given. Compared to [8] we still have a secret exponent d, but there is no public exponent e, so we will have to do something slightly different at some places. We will assume that there are l decryption servers, and a minimum of k of these are needed to make a correct decryption.

Key generation

Key generation starts out as in [8]: we find 2 primes p and q, that satisfies $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are primes and different from p and q. We set $n = pq$ and $m = p'q'$ and we make $g = (1+n)^j b^n \bmod n^2$

for a random $b \in Z_n^*$). In accordance with the basic idea from before we pick d to satisfy $d = 0 \bmod m$ and $d = j^{-1} \bmod n$. Now we make the polynomial $f(X) = \sum_{i=0}^{k-1} a_i X^i \bmod nm$, by picking a_i (for $0 < i < k$) as random values from $\{0, \cdots, n * m - 1\}$ and $a_0 = d$. The secret share of the i'th authority will be $s_i = f(i)$ for $1 \leq i \leq l$ and the public key will be (g, n). For verification of the actions of the decryption servers, we need the following fixed public values: v, generating the cyclic group of squares in $Z_{n^2}^*$ and for each decryption server a verification key $v_i = v^{\Delta s_i} \bmod n^2$, where $\Delta = l!$.

Encryption
To encrypt a message M, a random $r \in Z_n^*$ is picked and the cipher text is computed as $c = g^M r^n \bmod n^2$.

Share Decryption
The i'th authority will compute $c_i = c^{2\Delta s_i}$, where c is the ciphertext. Along with this will be a proof that $log_{c^4}(c_i^2) = log_v(v_i)$, which will convince us, that he has indeed raised to his secret exponent s_i [2]

Share combining
If we have the required k (or more) number of shares with a correct proof, we can combine them into the result by taking a subset S of k shares and combine them to

$$c' = \prod_{i \in S} c_i^{2\lambda_{0,i}^S} \bmod n^2 \qquad \text{where } \lambda_{0,i}^S = \Delta \prod_{i' \in S \setminus i} \frac{-i}{i - i'} \in Z$$

The value of c' will have the form $c' = c^{4\Delta^2 f(0)} = c^{4\Delta^2 d}$. Noting that $4\Delta^2 d = 0 \bmod \lambda$ and $4\Delta^2 d = 4\Delta^2 j^{-1} \bmod n$, we can conclude that $c' = (1+n)^{4\Delta^2 M} \bmod n^2$, where M is the desired plaintext, so this means we can compute $M = L(c')/4\Delta^2 \bmod n$.

Compared to the scheme proposed in [4], there are only minor differences: in [4], an extra random value related to the public element g is part of the public key and is used in the Share combining algorithm. This is avoided in our scheme by the way we choose d, and thus we get a slightly shorter public key and a slightly simpler decryption algorithm.

The system as described requires a trusted party to set up the keys. This may be acceptable as this is a once and for all operation, and the trusted party can delete all secret information as soon as the keys have been distributed. However, using multiparty computation techniques it is also possible to do the key generation without a trusted party. In the full version of this paper, we will include a proof in the random oracle model that this threshold version is as secure as Paillier's original scheme.

[2] A noninteractive proof for this using the Fiat-Shamir heuristic is easy to derive from the corresponding one in [8], see also [4].

3.2 Some Auxiliary Protocols

Suppose a prover P presents a sceptical verifier V with a ciphertext c and claims that it encodes plaintext i. A trivial way to convince V would be to reveal also the random choice r, then V can verify himself that $c = E(i, r)$. However, for use in the following, we need a solution where no extra useful information is revealed.

It is easy to see that that this is equivalent to convincing V that $cg^{-i} \bmod n^2$ is an n'th power. So we now propose a protocol for this which is a simple generalisation of the one from [5]. We note that this and the following protocols are not zero-knowledge as they stand, only honest verifier zero-knowledge. However, first zero-knowledge protocols for the same problems can be constructed from them using standard methods and secondly, in our applications, we will always be using them in a non-interactive variant based on the Fiat-Shamir heuristic, which means that we cannot obtain zero-knowledge, we can, however, obtain security in the random oracle model. As for soundness, we prove that the protocols satisfy so called special soundness (see [1]), which in particular implies that they satisfy standard knowledge soundness.

Protocol for n'th powers
Input: n, u
Private Input for P: v, such that $u = v^n \bmod n^2$

1. P chooses r at random mod n^2 and sends $a = r^n \bmod n^2$ to V
2. V chooses e, a random k bit number, and sends e to P.
3. P sends $z = rv^e \bmod n^2$ to V, and V checks that $z^n = au^e \bmod n^2$, and accepts if and only if this is the case.

It is now simple to show

Lemma 1. *The above protocol is complete, honest verifier zero-knowledge, and satisfies that from any pair of accepting conversations (between V and any prover) of form $(a, e, z), (a, e', z')$ with $e \neq e'$, one can efficiently compute an n'th root of u, provided 2^t is smaller than the smallest prime factor of n.*

Proof. Completeness is obvious from inspection of the protocol. For honest verifier simulation, the simulator chooses a random $z \in Z^*_{n^2}$, a random e, sets $a = z^n u^{-e} \bmod n^2$ and outputs (a, e, z). This is easily seen to be a perfect simulation.

For the last claim, observe that since the conversations are accepting, we have $z^n = au^e \bmod n^2$ and $z'^n = au^{e'} \bmod n^2$, so we get

$$(z/z')^n = u^{e-e'} \bmod n^2$$

Since $e - e'$ is prime to n by the assumption on 2^t, choose α, β such that $\alpha n + \beta(e - e') = 1$. Then let $v = u^\alpha (z/z')^\beta \bmod n^2$. We then get

$$v^n = u^{\alpha n}(z/z')^{n\beta} = u^{\alpha n} u^{\beta(e-e')} = u \bmod n^2$$

so that v is indeed the desired n'th root of u

In our application of this protocol, the modulus n will be chosen by a trusted party, or by a multiparty computation such that n has two prime factors of roughly the same size. Hence, if k is the bit length of n, we can set $t = k/2$ and be assured that a cheating prover can make the verifier accept with probability $\leq 2^{-t}$.

The lemma immediately implies, using the techniques from [1], that we can build an efficient proof that an encryption contains one of two given values, without revealing which one it is: given the encryption C and the two candidate plaintexts i_1, i_2, prover and verifier compute $u_1 = C/g^{i_1} \bmod n^2, u_2 = C/g^{i_2} \bmod n^2$, and the prover shows that either u_1 or u_2 is an n'th power. This can be done using the following protocol, where we assume without loss of generality that the prover knows an n'th root u_1, and where M denotes the honest-verifier simulator for the n-power protocol above:

Protocol 1-out-of-2 n'th power
Input: n, u_1, u_2
Private Input for P: v_1, such that $u_1 = v_1^n \bmod n^2$

1. P chooses r_1 at random mod n^2. He invokes M on input n, u_2 to get a conversation a_2, e_2, z_2. He sends $a_1 = r^n \bmod n^2, a_2$ to V
2. V chooses s, a random t bit number, and sends s to P.
3. P computes $e_1 = s - e_2 \bmod 2^k$ and $z_1 = r_1 v_1^{e_1} \bmod n^2$. He then sends e_1, z_1, e_2, z_2 to V.
4. V checks that $s = e_1 + e_2 \bmod 2^k$, $z_1^n = a_1 u_1^{e_1} \bmod n^2$ and $z_2^n = a_2 u_2^{e_2} \bmod n^2$, and accepts if and only if this is the case.

The proof techniques from [1] and Lemma 1 immediately imply

Lemma 2. *Protocol* 1-out-of-2 n'th power *is complete, honest verifier zero-knowledge, and satisfies that from any pair of accepting conversations (between V and any prover) of form $(a_1, a_2, s, e_1, z_1, e_2, z_2), (a_1, a_2, s', e_1', z_1', e_2', z_2')$ with $s \neq s'$, one can efficiently compute an n'th root of u_1, an n'th root of u_2, provided 2^t is less than the smallest prime factor of n.*

Our final building block allows a prover to convince a verifier that three encryptions contain values a, b and c such that $ab = c \bmod n$. For this, we propose a protocol inspired by a similar construction found in [2].

Protocol Multiplication-mod-n
Input: n, g, e_a, e_b, e_c
Private Input for P: a, b, c, r_a, r_b, r_c such that $ab = c \bmod n$ and $e_a = E(a, r_a), e_b = E(b, r_b), e_c = E(c, r_c)$

1. P chooses a random value $d \in Z_n$ and sends to V encryptions $e_d = E(d, r_d), e_{db} = E(db, r_{db})$.
2. V chooses e, a random t-bit number, and sends it to P.
3. P opens the encryption $e_a^e e_d = E(ea + d, r_a^e r_d \bmod n^2)$ by sending $f = ea + d \bmod n$ and $z_1 = r_a^e r_d \bmod n^2$. Finally, P opens the encryption $e_b^f (e_{db} e_c^e)^{-1} = E(0, r_b^f (r_{db} r_c^e)^{-1} \bmod n^2)$ by sending $z_2 = r_b^f (r_{db} r_c^e)^{-1} \bmod n^2$.

4. V verifies that the openings of encryptions in the previous step were correct, and accepts if and only if this was the case.

Lemma 3. *Protocol* Multiplication-mod-n *is complete, honest verifier zero-knowledge, and satisfies that from any pair of accepting conversations (between V and any prover) of form $(e_d, e_{db}, e, f, z_1, z_2), (e_d, e_{db}, e', f', z_1', z_2')$ with $e \neq e'$, one can efficiently compute the plaintext a, b, c corresponding to e_a, e_b, e_c such that $ab = c \bmod n$, provided 2^t is smaller than the smallest prime factor in n.*

Proof. Completeness is clear by inspection of the protocol. For honest verifier zero-knowledge, observe that the equations checked by V are $e_a^e e_d = E(f, z_1) \bmod n^2$ and $e_b^f (e_{db} e_c^e)^{-1} = E(0, z_2) \bmod n^2$. From this it is clear that we can generate a conversation by choosing first f, z_1, z_2, e at random, and then computing e_d, e_{db} that will satisfy the equations. This only requires inversion modulo n^2, and generates the right distribution because the values f, z_1, z_2, e are also independent and random in the real conversation. For the last claim, note first that since encryptions uniquely determine plaintexts, there are fixed values a, b, c, d contained in e_a, e_b, e_c, e_d, and a value x contained in e_{db}. The fact that the conversations given are accepting implies that $f = ea + d \bmod n$, $f' = e'a + d \bmod n$, $fb - x - ec = 0 = f'b - x - e'c \bmod n$. Putting this together, we obtain $(f - f')b = (e - e)'c \bmod n$ or $(e - e')ab = (e - e')c \bmod n$. Now, since $(e - e')$ is invertible modulo n by assumption on 2^t, we can conclude that $c = ab \bmod n$ (and also compute a, b and c).

The protocols from this section can be made non-interactive using the standard Fiat-Shamir heuristic of computing the challenge from the first message using a hash function. This can be proved secure in the random oracle model.

4 Efficient Electronic Voting

In [3], a general model for elections was used, which we briefly recall here: we have a set of voters $V_1, ..., V_M$, a bulletin board B, and a set of tallying authorities $A_1, ..., A_v$. The bulletin board is assumed to function as follows: every player can write to B, and a message cannot be deleted once it is written. All players can access all messages written, and can identify which player each message comes from. This ca n all be implemented in a secure way using an already existing public key infrastructure and server replication to prevent denial of service attacks. We assume that the purpose of the vote is to elect a winner among L candidates, and that each voter is allowed to vote for $t < L$ candidates.

In the following, h will denote a fixed hash function used to make non-interactive proofs according to the Fiat-Shamir heuristic. Alos, we will assume throughout that an instance of threshold version of Paillier's scheme with public key n, g has been set up, with the A_i's acting as decryption servers. We will assume that $n > M$, which will always be true for realistic values of these parameters.

The notation $Proof_P(S)$, where S is some logical statement will denote a bit string created by player P as follows: P selects the appropriate protocol from the previous section that can be used to interactively prove S. He computes the first message a in this protocol, computes $h(a, ID(P))$ where $ID(P)$ is his user identity in the system and, taking the result of this as the challenge from the verifier, computes the answer z. Then $Proof_P(S) = (a, z)$. The inclusion of $ID(P)$ in the input to h is done in order to prevent vote duplication.

A protocol for the case $L = 2$ is now simple to describe. This is equivalent to a yes/no vote and so each vote can thought of as a number equal to 0 for no and 1 for yes:

1. Each voter V_i decides on his vote v_i, he calculates $E_i = E(v_i, r_i)$, where r_i is randomly chosen. He also creates $Proof_{V_i}(E_i$ or E_i/g is an n'th power modulo $n^2)$ based on the 1-out-of-2 n'th power protocol. He writes the encrypted vote and proof to B.
2. Each A_j does the following: first set $E = 1$. Then for all i: check the proof written by V_i on B and if is it valid, then $E := E \cdot E_i \bmod n^2$. Finally, A_j executes his part of the threshold decryption protocol, using E as the input ciphertext, and writes his result to B.
3. From the messages written by the A_j's, anyone can now reconstruct the plaintext corresponding to E (possibly after discarding invalid messages). Assuming for simplicity that all votes are valid, it is evident that $E = \prod_i E(v_i, r_i) = E(\sum_i v_i \bmod n, \prod_i r_i \bmod n^2)$. So the decryption result is $\sum_i v_i \bmod n$ which is $\sum_i v_i$ since $n > M$.

Security of this protocol (in the random oracle model) follows easily from security of the sub-protocols used, and semantic security of Paillier's encryption scheme. Proofs will be included in the final version of this paper.

There are several ways to generalise this to $L > 2$. Probably the simplest way is to hold L parallel yes/no votes as above. A voter votes 1 for the candidates he wants, and 0 for the others. This means that V_i will send L votes of form

$$E_{ij} = E(v_{ij}, r_{ij}), Proof_{V_i}(E_{ij} \text{ or } E_{ij}/g \text{ is an } n\text{'th power modulo } n^2),\ j = 1, .., L$$

To prove that he voted for exactly t candidates, he also writes to B the number $\prod_j r_{ij} \bmod n^2$. This allows the talliers to verify that $\prod_j E(v_{ij}, r_{ij})$ is an encryption of t. This check is sufficient, since all individual votes are proved to be 0 or 1. It is immediate that decryption of the L results will immediately give the number of votes each candidate received.

We note that his easily generalises to cases where voters are allowed to vote for *up to* t candidates: one simply introduces t "dummy candidates" in addition to the actual L. We then execute the protocol as before, but with $t+L$ candidates. Each voter places the votes he does not want to use on dummy candidates.

The size of a vote in this protocol is seen to be $O(Lk)$, where k is the bit length of n, by simple inspection of the protocol. The protocol requires L decryption operations. As a numeric example, suppose we have $k = 1000, M = 64000, L = 64$. Then a vote in the above system has size about 80 Kbyte.

If the parameters are such that $L \log_2 M < k$ and $t = 1$, then we can do significantly better. These conditions will be satisfied in many realistic situations, such as for instance in the numeric example above. Note that for security reasons, one would want $k \geq 1000$ in any case, so the variant we propose here can indeed handle realistic values of M, L without having to increase k.

The basic idea is the following: a vote for candidate j, where $0 \leq j < L$ is defined to be an encryption of the number M^j. Each voter will create such an encryption and prove its correctness as detailed below. When all these encryptions are multiplied we get an encryption of a number of form $a = \sum_{j=0}^{L} a_j M^j \bmod n$, where a_j is the number of votes cast for candidate j. Since $L \log_2 M < k$, this relation also holds over the integers, so decrypting and writing a in M-ary notation will directly produce all the a_j's.

It remains to describe how to produce encryption hiding a number of form M^j, for some $0 \geq j < L$, and prove it was correctly formed. Let $b_0, ..., b_l$ be the bits in the binary representation of j, i.e. $j = b_0 2^0 + b_1 2^1 + ... + b_l 2^l$. Then clearly we have $M^j = (M^{2^0})^{b_0} \cdot ... \cdot (M^{2^l})^{b_l}$. Each factor in this product is either 1 or a power of M. This is used in the following algorithm for producing the desired proof (where P denotes the prover):

1. P computes encryptions $e_0, ..., e_l$ of $(M^{2^0})^{b_0}, ..., (M^{2^l})^{b_l}$. For each $i = 0...l$ he also computes $Proof_P(e_i/g \text{ or } e_i/g^{M^i} \text{ is an } n\text{'th power})$.
2. Let $F_i = (M^{2^0})^{b_0} \cdot ... \cdot (M^{2^i})^{b_i}$, for $i = 0...l$. P computes an encryption f_i of F_i, for $i = 1..l$. We set $f_0 = e_0$. Now, for $i = 1...l$, P computes

   $$Proof_P(\text{Plaintexts corr. to } f_{i-1}, e_i, f_i \text{ satisfy } F_{i-1} \cdot (M^{2^i})^{b_i} = F_i \bmod n\ ),$$

   based on the multiplication-mod-n protocol. The encryption f_l is the desired encryption, it can be verified from the e_i, f_i and all the proofs computed.

It is straightforward to see that a vote in this system will have length $O(k \log L)$ bits (still assuming, of course, that $k \log_2 M \leq k$).

With parameter values as in the numeric example before, a vote will have size about 15 Kbyte, a factor of more than 5 better than the previous system. Moreover, we need only 1 decryption operation as opposed to L before.

5 Reducing the Expansion Factor

Paillier's original system expands a plaintext to a ciphertext that is twice as long. We now show how this expansion factor can be made smaller. The basic idea is simple: to make keys, choose an RSA modulus n as usual, but do the computations modulo n^{s+1}, for some $s > 1$.

It is easy to see that the group $Z_{n^{s+1}}^*$ is isomorphic to the direct product of a cyclic group of order n^s and a group H of order $\phi(n)$. Again we will consider the factor group coming from factoring out H, which this time has order n^s. Carrying over ideas from the previous sections, we have the following lemma:

Lemma 4. *For any $s < p, q$, the element $n + 1$ has order n^s in $Z_{n^{s+1}}^*$.*

Proof. Consider the integer $(1+n)^i = \sum_{j=0}^{i} \binom{i}{j} n^j$. This number is 1 modulo n^{s+1} for some i if and only if $\sum_{j=1}^{i} \binom{i}{j} n^{j-1}$ is 0 modulo n^s. Clearly, this is the case if $i = n^s$, so it follows that the order of $1+n$ is a divisor in n^s, i.e., it is a number of form $p^\alpha q^\beta$, where $\alpha, \beta \leq s$. Set $a = p^\alpha q^\beta$, and consider a term $\binom{a}{j} n^{j-1}$ in the sum $\sum_{j=1}^{a} \binom{a}{j} n^{j-1}$. We claim that each such term is divisible by a: this is trivial if $j > s$, and for $j \leq s$, it follows because $j!$ can then not have p or q as prime factors, and so a must divide $\binom{a}{j}$. Now assume for contradiction that $a = p^\alpha q^\beta < n^s$. Without loss of generality, we can assume that this means $\alpha < s$. We know that n^s divides $\sum_{j=1}^{a} \binom{a}{j} n^{j-1}$. Dividing both numbers by a, we see that p must divide the number $\sum_{j=1}^{a} \binom{a}{j} n^{j-1}/a$. However, the first term in this sum after division by a is 1, and all the rest are divisible by n, so the number is in fact 1 modulo p, and we have a contradiction.

As previously, this means that we can number the residue classes of H in a natural way, and that for an element a in the i'th residue class, we have that $a = (1+n)^i h$ for an element $h \in H$, and so

$$a^\lambda = (1+n)^{i\lambda \bmod n^s} \bmod n^{s+1}$$

Just as before, this means that if we can compute i from $(1+n)^i \bmod n^{s+1}$ efficiently, then knowledge of λ will enable us to compute discrete logs efficiently in the factor group, and we can devise a decryption algorithm along the lines of the system using n^2 as modulus. Clearly, we have

$$L((1+n)^i \bmod n^{s+1}) = (i + \binom{i}{2} n + ... + \binom{i}{s} n^{s-1}) \bmod n^s$$

We now describe an algorithm for computing i from this number.

The general idea of the algorithm is to extract the value part by part, so that we first extract $i_1 = i \bmod n$, then $i_2 = i \bmod n^2$ and so forth. To see how this can be done let's look at the j'th step where we know i_{j-1}. This means that $i_j = i_{j-1} + k * n^{j-1}$ for some $0 \leq k < n$. If we use this in

$$L((1+n)^i \bmod n^{j+1}) = (i_j + \binom{i_j}{2} n + ... + \binom{i_j}{j} n^{j-1}) \bmod n^j$$

We can notice that each term $\binom{i_j}{t+1} n^t$ for $j > t > 0$ satisfies that $\binom{i_j}{t+1} n^t = \binom{i_{j-1}}{t+1} n^t \bmod n^j$. This is because the contributions from $k * n^{j-1}$ vanish modulo n^j after multiplication by n. This means that we get:

$$L((1+n)^i \bmod n^{j+1}) = (i_{j-1} + k * n^{j-1} + \binom{i_{j-1}}{2} n + ... + \binom{i_{j-1}}{j} n^{j-1}) \bmod n^j$$

Then we just rewrite that to get what we wanted

$$\begin{aligned} i_j &= i_{j-1} + k * n^{j-1} \\ &= i_{j-1} + L((1+n)^i \bmod n^{j+1}) - (i_{j-1} + \binom{i_{j-1}}{2} n \\ &\quad + \ldots + \binom{i_{j-1}}{j} n^{j-1}) \bmod n^j \\ &= L((1+n)^i \bmod n^{j+1}) - (\binom{i_{j-1}}{2} n + \ldots + \binom{i_{j-1}}{j} n^{j-1}) \bmod n^j \end{aligned}$$

This equation leads to the following algorithm:

```
i := 0;
for j:= 1 to s do
begin
  t1 := L(a mod n^(j+1));
  t2 := i;
  for k:= 2 to j do
  begin
    i := i − 1;
    t2 := t2 ∗ i mod n^j;
    t1 := t1 − (t2 ∗ n^(k−1))/k! mod n^j;
  end
  i := t1;
end
```

As in Paillier's original system, we will use as public key in our system pairs n, g, where n is an RSA modulus and g is a random element of order n^s in the factor group $Z^*_{n^{s+1}}/H$. The secret key is λ. An encryption of $i \in Z_{n^s}$ is of form $E_s(i, r) = g^i r^n \bmod n^{s+1}$, where r is randomly chosen in $Z^*_{n^{s+1}}$. By the algorithm above, knowledge of the secret λ enables us to compute discrete logarithms in the factor group, and a decryption algorithm follows easily.

If n is a k-bit number, then this system encrypts a ks bit message to a $k(s+1)$ bit ciphertext, and so the expansion factor can be brought arbitrarily close to 1, by choosing s larger.

It is worth noting that all the auxiliary protocols we developed earlier extend trivially to the case of computations modulo n^{s+1}, in particular the scheme is homomorphic w.r.t. addition of messages modulo n^s. Consequently our voting schemes can all be implemented based on this variant, which in particular gives the possibility in the most efficient scheme to accomodate a larger number of voters without increasing the size of the modulus.

The system is semantically secure, if the following assumption holds

Conjecture 1. Let A be any probabilistic polynomial time algorithm, and assume A gets n, g as input (n has k bits, n, g chosen as described above), it outputs

$i_0, i_1 \in Z_{n^s}$, it gets $E_s(i_b, r)$, where b is randomly chosen, and finally outputs a bit c. Let $p(A, k)$ be the probability that $c = b$. Then $\mid 1/2 - p(A, k) \mid$ is negligible in k.

By reducing everything modulo n^2, it is easily seen that this conjecture implies Paillier's original intractability assumption. It is not clear if the reverse implication is true, in particular, if i_0, i_1 are different modulo n^s but the same modulo n^2, then encryptions modulo n^{s+1} of i_0, i_1 might be easy to distinguish even if Paillier's original system is secure. We recommend that these systems are used with caution, until this family of intractability assumptions has been studied more carefully.

References

1. Cramer, Damgaård and Schoenmakers: *Proofs of partial knowlegde*, Proc. of Crypto 94, Springer Verlag LNCS series nr. 839.
2. R.Cramer, S.Dziembowski, I. Damgård, M.Hirt and T.Rabin: *Efficient Multiparty Computations Secure against an Adaptive Adversary*, Proc. of EuroCrypt 99, Springer Verlag LNCS series 1592, pp. 311-326.
3. R.Cramer, R.Gennaro, B.Schoenmakers: *A Secure and Optimally Efficient Multi-Authority Election Scheme*, Proceedings of EuroCrypt 97, Springer Verlag LNCS series, pp. 103-118.
4. P. Fouque, G. Poupard, J. Stern: *Sharing Decryption in the Context of Voting or Lotteries*, Proceedings of Financial Crypto 2000.
5. L. Guillou and J.-J. Quisquater: *A Practical Zero-Knowledge Protocol fitted to Security Microprocessor Minimizing both Transmission and Memory*, Proc. of EuroCrypt 88, Springer Verlag LNCS series.
6. M.Hirt and K.Sako: *Efficient Receipt-Free Voting based on Homomorphic Encryption*, to appear in Proc. of EuroCrypt 2000.
7. P.Pallier: *Public-Key Cryptosystems based on Composite Degree Residue Classes*, Proceedings of EuroCrypt 99, Springer Verlag LNCS series, pp. 223-238.
8. V.Shoup: *Practical Threshold Signatures*, to appear in Proceedings of EuroCrypt 2000.