A Generalization of Paillier's Public-Key System with Applications to Electronic Voting

Ivan Damgård, Mads Jurik and Jesper Buus Nielsen

Aarhus University, Dept. of Computer Science, **BRICS**[⋆]

Abstract. We propose a generalization of Paillier's probabilistic public key system, in which the expansion factor is reduced and which allows to adjust the block length of the scheme even after the public key has been fixed, without losing the homomorphic property. We show that the generalization is as secure as Paillier's original system and propose several ways to optimize implementations of both the generalized and the original scheme.

We construct a threshold variant of the generalized scheme as well as zero-knowledge protocols to show that a given ciphertext encrypts one of a set of given plaintexts, and protocols to verify multiplicative relations on plaintexts.

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 optimized such that for a certain range of the other parameter values, the ballotsize is logarithmic in L.

1 Introduction

In [18], 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 a generalization of Paillier's scheme using computations modulo n^{s+1}, for any $s \geq 1$. We also show that the system can be simplified (without degrading security) such that the public key can consist of only the modulus n. This allows instantiating the system such that the block length for the encryption can be chosen freely for each encryption, independently of the size of the public key, and without losing the homomorphic property. The generalization also allows reducing the expansion factor from 2 for Paillier's original system to almost 1. We prove that the generalization is as secure as Paillier's

[⋆] Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

original scheme. We also provide a number of ways to optimize both the encryption and decryption operations, in particular a new algorithm for encryption which, compared to a naive implementation of Paillier's original scheme, saves a factor of 4 in computing time. In general, it saves a factor of $4s$ compared to a straightforward implementation of the generalized system.

We propose a threshold variant of the generalized system, 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. We prove in the random oracle model that the scheme is as secure as a standard centralized implementation.

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 [8]. 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 [18] that since his encryption scheme is homomorphic, it may be applicable to electronic voting. In order to apply it in the framework of [8], 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.

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 [8]. However, the work needed to produce the result is reduced dramatically, as we now explain. With the El-Gamal encryption used in [8], 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 [8] 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. Moreover, we can scale easily to larger values of M and L without choosing new keys, just by going to a larger value of s.

We also give ways to 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. In each of these schemes, the size of a single ballot is $O(k \cdot L)$, where k is the bit length of the modulus used[1]. We propose a variant using a different technique where ballots have size $O(max(k, L\log M) \cdot \log L)$. Thus for $k \geq L\log M$, this is much more efficient, and even 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 $L > 2$.

Some of the results in this paper were presented in preliminary form in [9].

2 Related Work

In work independent from, but earlier than ours, Fouque, Poupard and Stern [12] proposed the first threshold version of Paillier's original scheme. Like our threshold scheme, [12] uses an adaptation of Shoup's threshold RSA scheme [19], but beyond this the techniques are somewhat different, in particular because we construct a threshold version for our generalized crypto system (and not only Paillier's original scheme). In [12] voting was also pointed out as a potential application, however, no suggestion was made there for protocols to prove that an encrypted vote is correctly formed, something that is of course necessary for a secure election in practice.

In work done concurrently with and independent from ours, Baudron, Fouque, Pointcheval, Poupard and Stern [3] propose a voting scheme somewhat similar to ours. Their work can be seen as being complementary to ours in the sense that their proposal is more oriented toward the system architectural aspects of a large scale election, and less toward optimization of the building blocks. To compare to their scheme, we first note that there the modulus length k must be chosen such that $k - 1 > L\log M$. The scheme produces ballots of size $O(k \cdot L)$. An estimate with explicit constants is given in [3] in which the dominating term in our notation is $9kL$.

Because our voting scheme uses the generalized Paillier crypto system, k can be chosen independently from L, M, in particular the scheme can scale to any size of election, even after the keys have been generated. But if we choose k as in [3], i.e. $k - 1 > L\log M$, then the ballots we produce have size $O(k \cdot \log L)$. Working out the concrete constants involved, one finds that our complexity is

[1] All complexities given here assume that the length of challenges for the zero-knowledge proofs is at most k.

dominated by the term $10k \log L$. So already for moderate size elections we have gained a significant factor in complexity compared to [3].

In [16], 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 [8]. It can also be applied to our scheme, with the same efficiency gain as in the non-receipt free case.

When using the threshold version of our scheme, we assume for simplicity a trusted dealer for setting up the keys initially, and we assume that the modulus used is a safe prime product, similar to what is done in Shoup's paper [19]. In [10], Damgård and Koprowski propose techniques by which one can drop these restrictions from Shoup's scheme, at the expense of an extra intractability assumption. The same idea can be easily applied to our scheme thus producing a scheme without a trusted dealer and using a general RSA modulus. The threshold version of our scheme can also be used for *general* secure multiparty computation as shown by Cramer, Damgård and Nielsen in [4].

3 A Generalization of Paillier's Probabilistic Encryption Scheme

The public-key crypto system we describe here uses computations modulo n^{s+1} where n is an RSA modulus and s is a natural number. It contains Paillier's scheme [18] as a special case by setting $s = 1$.

Consider a modulus $n = pq$, p, q odd primes, where $gcd(n, \phi(n)) = 1$. When p, q are large and randomly chosen, this will be satisfied except with negligible probability. Such an n will be called *admissible* in the following. For such an n, $Z^*_{n^{s+1}}$ as a multiplicative group is a direct product $G \times H$, where G is cyclic of order n^s and H is isomorphic to Z^*_n. This follows directly from the Chinese remainder theorem and the fact that $Z^*_{p^{s+1}}$ is cyclic of order $(p-1)p^s$. Thus, the factor group $\bar{G} = Z^*_{n^{s+1}}/H$ is also cyclic of order n^s. For an arbitrary element $a \in Z^*_{n^{s+1}}$, we let $\bar{a} = aH$ denote the element represented by a in the factor group $\bar{G}$.

Lemma 1. *For any admissible n and $s < p, q$, 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 p, so the number is in fact 1 modulo p, and we have a contradiction.

Since then the order of H is relatively prime to n^s the above lemma implies immediately that the element $\overline{1+n} := (1+n)H \in \bar{G}$ is a generator of $\bar{G}$, except possibly for $s \geq p, q$. So the cosets of H in $Z^*_{n^{s+1}}$ are

$$H, (1+n)H, (1+n)^2 H, ..., (1+n)^{n^s - 1} H,$$

which leads to a natural numbering of these cosets. The following lemma captures the structure of $Z^*_{n^{s+1}}$ in a more concrete way:

Lemma 2. *For any admissible n and $s < p, q$, the map $\psi_s : Z_{n^s} \times Z^*_n \to Z^*_{n^{s+1}}$ given by $(x, r) \mapsto (1+n)^x r^{n^s} \bmod n^{s+1}$ is an isomorphism, where $\psi_s(x_1 + x_2 \bmod n^s, r_1 r_2 \bmod n) = \psi_s(x_1, r_1)\psi_s(x_2, r_2) \bmod n^{s+1}$.*

Proof. Let $\pi : Z^*_n \to Z^*_{n^{s+1}}$ be given by $r \mapsto \psi_s(0, r) = r^{n^s} \bmod n^{s+1}$. By the above enumeration of the cosets of H it is enough to prove that $\pi(r_1 r_2 \bmod n) = \pi(r_1)\pi(r_2) \bmod n^{s+1}$ and that π maps Z^*_n injectively to H. First, it is clear that $\pi(r) \in H$. By looking at the binomial expansion it is easy to see that $r^{n^s} \equiv (r+n)^{n^s} \pmod{n^{s+1}}$. This proves the homomorphic property directly and by the pigeon hole principle implies that π is injective.

This lemma gives us the following encoding of the cosets: $(\overline{1+n})^i = \psi_s(i, Z^*_n)$.

The final technical observation we need is that ψ can be inverted given the factorization of n. In particular, taking discrete logarithms base $\overline{n+1}$ in $\bar{G}$ is easy given the factorization.

Theorem 1. *For any admissible n and $s < p, q$, the map $\psi_s : Z_{n^s} \times Z^*_n \to Z^*_{n^{s+1}}$ given by $(x, r) \mapsto (1+n)^x r^{n^s} \bmod n^{s+1}$ can be inverted in polynomial time given $\lambda(n)$, the least common multiple of $p-1$ and $q-1$.*

Proof. We first show how to find i from $(1+n)^i \bmod n^{s+1}$. If we define the function $L()$ by $L(b) = (b-1)/n$ then 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. It is easy to extract $i_1 = L((1+n)^i \bmod n^2) = i \bmod n$. Now we can extract the rest by the following induction step: In the j'th step 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

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

This equation leads to the following algorithm:

$i := 0$;
for j:= 1 **to** s **do**
begin
  $t_1 := L(a \bmod n^{j+1})$;
  $t_2 := i$;
  for k:= 2 **to** j **do**
  begin
    $i := i - 1$;
    $t_2 := t_2 * i \bmod n^j$;
    $t_1 := t_1 - \frac{t_2 * n^{k-1}}{k!} \bmod n^j$;
  end
  $i := t_1$;
end

Assume now that we are given $c = (1+n)^i r^{n^s} \bmod n^{s+1}$. We show how to find i and r given λ. To find i compute

$$c^\lambda = (1+n)^{i\lambda \bmod n^s} r^{n^s \lambda \bmod n^s \lambda} = (1+n)^{i\lambda \bmod n^s} \ .$$

Then using the above algorithm find $i\lambda \bmod n^s$ and extract i. Now compute $r^{n^s} = c(1+n)^{-i} \bmod n^{s+1}$ and compute a such that $a\lambda + 1 = 0 \bmod n^s$. This is possible because $\gcd(\lambda, n^s) = 1$. Then

$$(r^{n^s})^{\frac{a\lambda+1}{n^s}} \bmod n = r^{a\lambda+1} \bmod n = (r^\lambda)^a r \bmod n = r \bmod n = r \ .$$

We are now ready to describe our crypto system. In fact, for each natural number s, we can build a crypto system CS_s, as follows:

Key Generation On input the security parameter k, choose an admissible RSA modulus $n = pq$ of length k bits[2]. Also choose an element $g \in Z^*_{n^{s+1}}$ such that $g = (1+n)^j x \bmod n^{s+1}$ for a known j relatively prime to n and $x \in H$. This can be done, e.g., by choosing j, x at random first and computing g; some alternatives are described later. Let λ be the least common multiple of $p-1$ and $q-1$. By the Chinese Remainder Theorem, choose d such that $d \bmod n \in Z^*_n$ and $d = 0 \bmod \lambda$. Any such choice of d will work in the following. In Paillier's original scheme $d = \lambda$ was used, which is the smallest possible value. However, when making a threshold variant, other choices are better - we expand on this in the following section.
Now the public key is n, g while the secret key is d.

Encryption The plaintext set is Z_{n^s}. Given a plaintext i, choose a random $r \in Z^*_n$, and let the ciphertext be $E(i,r) = g^i r^{n^s} \bmod n^{s+1}$.

Decryption Given a ciphertext c, first compute $c^d \bmod n^{s+1}$. Clearly, if $c = E(i,r)$, we get

$$
\begin{aligned}
c^d &= (g^i r^{n^s})^d = ((1+n)^{ji} x^i r^{n^s})^d = (1+n)^{jid \bmod n^s} (x^i r^{n^s})^{d \bmod \lambda} \\
&= (1+n)^{jid \bmod n^s}
\end{aligned}
$$

Now apply the algorithm from the proof of Theorem 1 to compute $jid \bmod n^s$. Applying the same method with c replaced by g clearly produces the value $jd \bmod n^s$, so this can either be computed on the fly or be saved as part of the secret key. In any case we obtain the cleartext by $(jid) \cdot (jd)^{-1} = i \bmod n^s$.

Clearly, this system is additively homomorphic over Z_{n^s}, that is, the product of encryptions of messages i, i' is an encryption of $i + i' \bmod n^s$.

To facilitate comparison with Paillier's original system, we have kept the above system description as close as possible to that of Paillier. In particular, the description allows choosing g in a variety of ways. However, as we shall see, semantic security of the system is equivalent to a particular computational assumption, no matter how we choose g, in particular we may as well simplify matters and choose $g = n+1$ always. This also allows a more efficient implementation. Therefore, in the following sections, when we refer to CS_s, we usually mean the above system with $g = n+1$.

3.1 Security

There are two basic flavors or strengths of security that one may consider, namely

[2] Strictly speaking, we also need that $s < p, q$, but this is insignificant since in practice, s will always be much smaller than p, q

– Is the scheme one-way, i.e., is it hard to compute the plaintext from the ciphertext?
– Is the scheme semantically secure, i.e., does any information at all about the plaintext leak, given the ciphertext?

We give first a short informal discussion on one-wayness, and then look at semantic security in more detail.

The homomorphic property of the scheme means that the problem of computing the plaintext from the ciphertext (and the public key) is random self-reducible: given any ciphertext c and public key n, g, one may choose $i \in Z_{n^s}$, $j \in Z^*_{n^s}$, $r, r' \in Z^*_n$ at random and try to decrypt the ciphertext $c' = cg^i r^{n^s} \bmod n^{s+1}$ with respect to public key n, g' where $g' = g^j r'^{n^s} \bmod n^{s+1}$. If this succeeds, one can find the original plaintext by multiplying by j and subtracting i modulo n^s. Note that c', g' is a random ciphertext-generator pair, no matter how c, g were chosen. So any algorithm that can break a non-trivial fraction of the ciphertexts and choices of g can also break a random instance with significant probability. This motivates calling our scheme one-way if it is hard to find the plaintext given a random public key n, g and a random ciphertext c.

We have

Proposition 1. *If for some t the scheme CS_t is one-way, then CS_s is one-way for any $s > t$. Especially CS_s is one-way for any s if Paillier's original scheme CS_1 is one-way.*

Proof. Assume that $s > t$ and that CS_t is one-way. Assume for the sake of contradiction that CS_s is not one-way. Then given a public key n, g and a ciphertext c_t from CS_t, we can transform this to a decryption problem in CS_s instead. Concretely this means we consider c_t as a number modulo n^{s+1} (instead of n^{t+1}), and choose as the public generator a random number $\tilde{g} \in Z^*_{n^{s+1}}$, such that $\tilde{g} \bmod n^{t+1} = g$. We then randomize c_t (modulo n^{s+1}) as described above. This produces a random instance of the decryption problem in CS_s, so by assumption we can find the plaintext m in CS_s corresponding to c_t. We have of course that $m \in Z_{n^s}$, and now clearly $m \bmod n^t$ is the plaintext corresponding to c_t in CS_t, so that CS_t is not one-way either.

If we want to claim that a cryptosystem "hides" the plaintext in any reasonable sense, the one-way assumption is essentially the weakest possible assumption one can make. In [7], Catalano, Gennaro and Howgrave-Graham show that this assumption for CS_1 implies that one can make a *semantically secure* system hiding a logarithmic number of bits per ciphertext in the original system, and that a somewhat stronger assumption implies a system hiding a linear number of bits per ciphertext. It is easy to generalize these results to CS_s. None of the schemes constructed this way will be homomorphic, however.

The semantic security of our schemes can be based on the following assumption, introduced by Paillier in [18], the *decisional composite residuosity assumption* (DCRA):

Conjecture 1. Let A be any probabilistic polynomial time algorithm, and assume A gets n, x as input. Here n has k bits, and is chosen as described above, and x is either random in $Z^*_{n^2}$ or it is a random n'th power in $Z^*_{n^2}$ (that is, a random element in the subgroup H defined earlier). A outputs a bit b. Let $p_0(A, k)$ be the probability that $b = 1$ if x is random in $Z^*_{n^2}$ and $p_1(A, k)$ the probability that $b = 1$ if x is a random n'th power. Then $\mid p_0(A, k) - p_1(A, k) \mid$ is negligible in k.

Here, "negligible in k" as usual means smaller than $1/f(k)$ for any polynomial $f()$ and all large enough k.

We now discuss the semantic security of CS_s. There are several equivalent formulations of semantic security. We will use the following:

Definition 1. *An adversary A against a public-key crypto system gets the public key pk generated from security parameter k as input and outputs a message m. Then A is given an encryption under pk of either m or a message chosen uniformly in the message space, and outputs a bit. Let $p_0(A, k)$, respectively $p_1(A, k)$ be the probability that A outputs 1 when given an encryption of m, respectively a random encryption. Define the* advantage *of A to be $Adv(A, k) = |p_0(A, k) - p_1(A, k)|$. The crypto system is* semantically secure *if for any probabilistic polynomial time adversary A, $Adv(A, k)$ is negligible in k.*

In [18], Paillier showed that his crypto system (which is equivalent in security to our CS_1) is semantically secure if and only if DCRA holds. This holds for any choice of g, and follows easily from the fact that given a ciphertext c that is either random or encrypts a message i, we have that $cg^{-i} \bmod n^2$ is either random in $Z^*_{n^2}$ or a random n'th power. In particular, assuming DCRA, one may choose $g = n + 1$ always without degrading security. We now show that security of CS_s is equivalent to DCRA:

Theorem 2. *For any s, the crypto system CS_s is semantically secure if and only if the DCRA is true. This holds even if s is allowed to increase polynomially in the security parameter.*

Proof. From a ciphertext in CS_s, one can obtain a ciphertext in CS_1 by reducing modulo n^2, this implicitly reduces the message modulo n. It is therefore clear that if DCRA fails, then CS_s cannot be secure for any s.

For the converse, we assume that CS_s is not secure and we start by showing a relation between security of CS_s and that of CS_t for values of $t < s$.

The message space of CS_s is Z_{n^s}. Thus any message m can be written in n-adic notation as an s-tuple $(m_s, m_{s-1}, ..., m_1)$, where each $m_i \in Z_n$ and $m = \sum_{i=0}^{s-1} m_{i+1} n^i$. Let $D_n(m_s, ..., m_1)$ be the distribution obtained by encrypting the message $(m_s, ..., m_1)$ under public key n. If one or more of the m_i are replaced by $*$'s, this means that the corresponding position in the message is chosen uniformly in Z_n before encrypting.

Now, assume for simplicity that s is even, consider any adversary A against CS_s, and assume that $Adv(A, k) \geq 1/f(k)$ for some polynomial $f()$ and infinitely many values of k. For any such value, we can assume without loss of generality, that we have $p_0(A, k) - p_1(A, k) \geq 1/f(k)$. Suppose we make a public key n from security parameter k, show it to A, get a message $(m_s, ..., m_1)$ from A and show A a sample of $D_n(*, ..., *, m_{s/2}, ..., m_1)$. Let $q(A, k)$ be the probability that A now outputs 1. Of course, we must have

$$(*) \qquad p_0(A, k) - q(A, k) \geq \frac{1}{2f(k)} \quad \text{or} \quad q(A, k) - p_1(A, k) \geq \frac{1}{2f(k)}$$

and one of these cases must be true for infinitely many values of k. In the first case in $(*)$, we can make a successful adversary against $CS_{s/2}$, as follows: we get the public key n, show it to A, get $(m_s, ..., m_1)$, and return $(m_s, ..., m_{1+s/2})$ as output. We will get a ciphertext c that either encrypts $(m_s, ..., m_{1+s/2})$ in $CS_{s/2}$, or is a random ciphertext. If we consider c as an element in $Z^*_{n^{s+1}}$, we know it is an encryption of some plaintext, which must have either $(m_s, ..., m_{1+s/2})$ or $s/2$ random elements in its least significant positions. Hence $c^{n^{s/2}} \bmod n^{s+1}$ is an encryption of $(m_s, ..., m_{1+s/2}, 0, ..., 0)$ or $(*, .., *, 0, ..., 0)$. We then make a random encryption d of $(0, .., 0, m_{s/2}, ..., m_1)$, give $c^{n^{s-1}} d \bmod n^{s+1}$ to A and return the bit A outputs. Now, if c encrypts $(m_s, ..., m_{1+s/2})$, we have shown to A a sample of $D_n(m_s, ..., m_1)$, and otherwise a sample of $D_n(*, m_{s/2}, ..., m_1)$. So by assumption on A, this breaks CS_1 with an advantage of $1/2f(k)$ for infinitely many k.

In the second case of $(*)$, we can also make an adversary against $CS_{s/2}$, as follows: we get the public key n, show it to A, and get a message $(m_s, ..., m_1)$. We output $(m_{s/2}, ..., m_1)$ and get back a ciphertext c that encrypts in $CS_{s/2}$ either $(m_{s/2}, ..., m_1)$ or something random. If we consider c as a number modulo n^{s+1}, we know that the corresponding plaintext in CS_s has either $(m_{s/2}, ..., m_1)$ or random elements in the least significant $s/2$ positions - and something unknown in the top positions. We make a random encryption d of $(*, .., *, 0, ..., 0)$, show $cd \bmod n^{s+1}$ to A and return the bit A outputs. If c encrypted $(m_{s/2}, ..., m_1)$, we have shown A a sample from $D_n(*, .., *, m_{s/2}, ..., m_1)$, and otherwise a sample from $D_n(*, ..., *)$. So again this breaks $CS_{s/2}$ with an advantage of $1/2f(k)$ for infinitely many k.

To sum up, we have: for any adversary A against CS_s, s even, there exists an adversary A' against $CS_{s/2}$, such that $Adv(A', k) \geq 1/2f(k)$ for infinitely many k. Similarly, for odd s, we can show existence of an adversary against either $CS_{(s+1)/2}$ or $CS_{(s-1)/2}$ with advantage at least $1/2f(k)$ for infinitely many k.

Repeated use of this result shows that for any adversary A against CS_s, there exists an adversary against CS_1 with advantage at least $1/2sf(k)$ for infinitely many k. Thus, since s is polynomially bounded as a function of k, CS_1 is not semantically secure, and this contradicts Paillier's original result.

From the point of view of exact security analysis, one can note that from the proof above, it follows that the maximal advantage with which CS_s can be broken is at most a factor of $2s$ larger than the corresponding advantage for CS_1. Thus, there is no great security risk in using large values of s, if one believes that CS_1 is secure in the first place.

3.2 Adjusting the Block length

As mentioned, we may choose $g = n + 1$ always without losing security, and the public key may then consist only of the modulus n. This means that we can decide on a value for s at any point after the keys have been generated, or even let the sender decide on the fly when he encrypts a message. Concretely, the system will then work as follows:

Key Generation Choose an admissible RSA modulus $n = pq$. Now the public key is n while the secret key is λ, the least common multiple of $(p-1)$ and $(q-1)$.

Encryption Given a plaintext i represented as a non-negative integer, choose a s such that $i < n^s$, choose a random $r \in Z_n^*$, and let the ciphertext be $E(i,r) = (1+n)^i r^{n^s} \bmod n^{s+1}$.

Decryption Given a ciphertext c, compute $c^\lambda \bmod n^{s+1}$ (note that from the length of c, one can decide the correct value of s except with negligible probability). Clearly, if $c = E(i,r)$, we get

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

Now apply the algorithm from Theorem 1 to compute $i\lambda \bmod n^s$ and get the message by multiplying by λ^{-1} modulo n^s.

It is an immediate corollary to Proposition 1 and Theorem 2 that the above scheme is one-way if CS_1 is one-way respectively is semantically secure if the DCRA holds.

4 Some Optimizations and Implementation Issues

4.1 An Alternative Encryption Function

Let $\psi_s : Z_{n^s} \times Z_n^* \to Z_{n^{s+1}}^*$ be the isomorphism given by $(x,r) \mapsto (1+n)^x r^{n^s} \bmod n^{s+1}$ in Lemma 2. In the above we encrypt an element $i \in Z_{n^s}$ by a random element from the coset $\overline{(1+n)}^i = \psi_s(i, Z_n^*)$. This element is chosen as $c = \psi_s(i,r)$ for random $r \in Z_n^*$. Note that if we reduce a ciphertext modulo n, we obtain:

$$c \bmod n = (1+n)^x r^{n^s} \bmod n = r^{n^s} \bmod n$$

The Jacobi symbol modulo n is easy to compute, even without the factors (see e.g. [2]), and since n^s is odd and the Jacobi symbol is multiplicative, we see that from $c = \psi_s(i,r)$, we can compute the Jacobi symbol of r efficiently. Further, by multiplying c by a number of form $\psi_s(0,\tilde{r})$, where $\tilde{r}$ is an arbitrary constant with the same Jacobi symbol as r, we obtain a ciphertext $c' = \psi_s(i,r') = \psi_s(i, r\tilde{r})$, where r' is guaranteed to have Jacobi symbol 1. It easily follows that the crypto system which is like CS_s, except that we restrict r to have Jacobi symbol 1, is exactly as secure as CS_s, under any notion of security. We now exploit this to obtain an alternative and more efficient encryption function.

Using standard techniques we can generate a random RSA modulus $n = pq$ with known p and q such that $p = 3 \bmod 4$, $q = 3 \bmod 4$, $\gcd(p-1, q-1) = 2$. This means that the subgroup of quadratic residues $SQ(n)$ is cyclic and has odd order, say α. We can also ensure that all elements in this subgroup - except for a negligible fraction - are generators. This can be done by picking p, q such that all prime factors in $p-1, q-1$ except 2 are sufficiently large. One extreme special case of this is when n is a safe prime product, which is an option we use later for the threshold version of the scheme.

Let $Z_n^*[+]$ be the elements with Jacobi symbol 1 in Z_n^*. We have that $Z_n^*[+]$ contains $SQ(n)$, has order 2α and is also cyclic. Finally, $-1 \in Z_n^*[+] \setminus SQ(n)$ by choice of n.

All this implies that if we choose at random $x \in Z_n^*$ and let $h = -x^2 \bmod n$ then, except with negligible probability, $\langle h \rangle = Z_n^*[+]$. This then allows us to generate a uniformly random element r from $Z_n^*[+]$ as $h^a \bmod n$, where a is a uniformly random integer from $[0, (p-1)(q-1)/2)$. However, since $(p-1)(q-1)/2$ is the secret key, this would allow only the owner of the secret key to encrypt, which would of course be useless. We can remedy this by using a result from [13]. Let (n, h) be generated as above. Let a be a uniformly random integer from $[0, (p-1)(q-1)/2)$ and let a' be a uniformly random element from $[0, 2^{\lceil k/2 \rceil})$. Then by [13, Theorem 3.2] the random variables $(n, h, h^a \bmod n)$ and $(n, h, h^{a'} \bmod n)$ are computationally indistinguishable assuming the intractability of factoring, which is implied by the DCRA. This means that even though $h^{a'} \bmod n$ is not a uniformly random element from $Z_n^*[+]$, it cannot be distinguished from a uniformly random element from $Z_n^*[+]$ by any polynomial time algorithm, which suffices for our application. This gives us the following crypto system $\widetilde{CS}_s$.

Key Generation Choose an admissible RSA modulus $n = pq$ of length k bits, where $p \equiv q \equiv 3 \pmod 4$, $\gcd(p-1, q-1) = 2$, and such that a random square generates $SQ(n)$ except with negligible probability. Choose a generator h of $Z_n[+]$ as described above. Now the public key is (n, h) while the secret key is $\lambda = (p-1)(q-1)/2$, the least common multiple of $(p-1)$ and $(q-1)$.

Encryption Given a plaintext $i \in Z_{n^s}$ choose a random $a \in Z_{2^{\lceil k/2 \rceil}}$ and let the ciphertext be $\tilde{E}(i, a) = (1+n)^i (h^a \bmod n)^{n^s} \bmod n^{s+1} = E(i, h^a \bmod n)$.

Decryption As before.

The following theorem follows directly from the fact that $h^a \bmod n$ is pseudorandom in Z_n^* under DCRA, that h can be generated given just n, and that the security of CS_s is unchanged when restricting the randomness to Jacobi symbol 1.

Theorem 3. *For any s, the crypto system $\widetilde{CS}_s$ is semantically secure if and only if the DCRA is true. This holds even if s is allowed to increase polynomially in the security parameter.*

From an exact security point of view, one should be aware that in order to argue the security, we are using the DCRA twice: first to argue that $\widetilde{CS}_s$ is as secure as CS_s (namely $h^a \bmod n$ is pseudorandom) and then to argue that CS_s is secure. This means that if we want to build instances of $\widetilde{CS}_s$ that can be broken with advantage no larger than instances of CS_s with security parameter k, we need to use moduli that are somewhat longer than k bits. How much longer depends on exactly how strong assumptions we are willing to make, and on the complexity of the reduction in the result of [13]. This may partly eliminate the efficiency advantage we show below for $\widetilde{CS}_s$. On the other hand, this issue can be completely avoided by using more randomness, namely we choose a as a random number modulo $n/2$, instead of a random $k/2$-bit number. Then $h^a \bmod n$ will be statistically close to a random element in $\langle h \rangle$, without any assumptions, and up to a negligible term, we have the same security for this variant of $\widetilde{CS}_s$ as for CS_s. This will only cost a factor of 2 in performance of the encryption.

4.2 Optimizations of Encryption and Decryption

Encryption. While encrypting, instead of computing $(1+n)^m$ directly, we can can compute it according to:

$$(1+n)^m = 1 + mn + \binom{m}{2} n^2 + ... + \binom{m}{s} n^s \bmod n^{s+1}$$

this trades an exponentiation with a $O(n^s)$ size exponent for $O(s)$ multiplications by calculating the binomials using:

$$\binom{m}{j} = \binom{m}{j-1} \frac{m-j+1}{j}$$

In the i'th step we calculate $\binom{m}{i} n^i \bmod n^{s+1}$, and since there is a multiplication by n^i it is enough to calculate the binomial modulo n^{s-i+1}. To further optimize the computations the $(j!)^{-1} n^j$ can be precomputed. The pseudo algorithm for calculating the $(n+1)^m$ part of the encryption looks like this (where precomp is an array of the precomputed values, $precomp[j] := (j!)^{-1} n^j \bmod n^{s+1}$):

$c := 1 + mn;$
$tmp := m;$
for j:= 2 **to** s **do**
begin
  $tmp := tmp \cdot (m - j + 1) \bmod n^{s-j+1};$
  $c := c + tmp \cdot precomp[j] \bmod n^{s+1};$
end

In the crypto system $\widetilde{CS}_s$ the elements from H is generated as $(h^a \bmod n)^{n^s} \bmod n^{s+1}$ which, if computed naively, certainly leads to no optimization. However, a simple observation allows us to reduce the number of steps used in this computation. Let $h_s = h^{n^s} \bmod n^{s+1}$. Then using the isomorphism from Theorem 1, we have

$$(h^a \bmod n)^{n^s} \bmod n^{s+1} = \psi_s(0, h^a \bmod n) = \psi_s(0, h)^a = h_s^a \bmod n^{s+1}$$

It follows that $\tilde{E}(i, a) = (1+n)^i h_s^a \bmod n^{s+1}$. If we precompute and save h_s, then using standard methods for exponentiation with a fixed base, $h_s^a \bmod n^{s+1}$ can be computed by an expected number of $k/4$ multiplications modulo n^{s+1} and hence the entire encryption can be done in $k/4 + 2s$ multiplications. Compared to a straightforward implementation of CS_s with the same k value, where 2 full scale exponentiations are made, this saves a factor of about $4s$ in computing time, and in particular this is four times as fast as Paillier's original system.

Decryption. The technique of precomputing factors in binomial coefficients to make encryption faster also applies to the corresponding computations in decryption (see the algorithm in the proof of Theorem 1). Also in the same way as with encryption, we can exploit the fact that the algorithm involves modular multiplication of a variable by a power of n, which means the value of that variable only needs to be known modulo a smaller power of n.

Another thing that can be optimized is the use of the L function. In the algorithm from Theorem 1 the L function is calculated once per iteration of the for-loop. Instead of doing this we can calculate the largest of these: $L(a \bmod n^{s+1})$ and use the property that $L(a \bmod n^{j+1}) = L(a \bmod n^{s+1}) \bmod n^j$. This means that we can remove all but 1 of the division and the modular reductions we make are smaller.

The standard trick of splitting the computations up and doing them modulo relatively prime parts of the modulus can also be used here with the moduli p^j and q^j in the j'th run of the outer loop. One should be aware though that we need to use different L functions for p and q, namely $L_q(a) = ((a - 1 \bmod q^{s+1})/q) \cdot p^{-1} \bmod q^s$ and $L_p(a) = ((a - 1 \bmod p^{s+1})/p) \cdot q^{-1} \bmod p^s$.

In this case, decryption can be speeded up by precomputing p^j, q^j for $1 \leq j \leq s$, and the $n^{k-1}k!^{-1} \bmod p^j$, $n^{k-1}k!^{-1} \bmod q^j$ for $2 \leq k \leq j \leq s$.

Performance Evaluations. In figure 1 and 2 the generalized crypto system is compared to the El-Gamal and RSA crypto systems. The table is focused on a fixed plaintext size and variable size of security parameter for the generalized cryptosystem. This comparison corresponds to a scenario where you need a certain fixed plaintext size (for instance a large scale election) and it might be sufficient with a smaller security parameter. It shows that if the security parameter doesn't need to have the same size as the encryption block then a significant performance improvement can be achieved.

In figure 3 there is a comparison with the number of milli-seconds it takes to encrypt a bit using same security parameter, but a variable block size. It shows that using El-Gamal and the generalized crypto system achieves almost the same rates of encryption. It also shows - as expected - that the encryption time per bit increases somewhat with larger s values. Thus, if small ciphertext expansion and large block size is important, this can be achieved at a reasonable performance penalty; but if speed is the only important parameter, $s = 1$ is the best choice.

Fig. 1. Comparison with 2048 bit plaintext size, using java implementation

	El-Gamal	Generalized Paillier		RSA
		$s = 1$	$s = 2$	
Security	2048	2048	1024	2048
Ciphertext size	4096	4096	3072	2048
Expansion factor	2	2	1.5	1
Encryption (ms)	1980	1969	578	8
Decryption (ms)	996	1030	312	272

Fig. 2. Comparison with 4096 bit plaintext size, using java implementation

	El-Gamal	Generalized Paillier				RSA
		$s = 1$	$s = 2$	$s = 3$	$s = 4$	
Security	4096	4096	2048	1366	1024	4096
Ciphertext size	8192	8192	6144	5462	5120	4096
Expansion factor	2	2	1.5	1.33	1.25	1
Encryption (ms)	15205	15264	4397	2370	1591	32
Decryption (ms)	7611	7779	2290	1281	873	2001

Fig. 3. ms per bit encrypted/decrypted on a 750 MHz Pentium III using java implementation

Security parameter	El-Gamal	Generalized Paillier			RSA
		$s = 1$	$s = 2$	$s = 4$	
Encryption					
1024	0.264	0.262	0.284	0.387	0.002
2048	0.967	0.955	1.067	1.480	0.004
4096	3.711	3.705	4.146	5.755	0.008
8192	14.467	14.507	16.244	22.617	0.015
Decryption					
1024	0.132	0.149	0.153	0.214	0.039
2048	0.489	0.503	0.559	0.780	0.132
4096	1.865	1.898	2.128	2.958	0.486
8192	7.286	7.349	8.244	11.461	1.854

5 Some Building Blocks

5.1 A Threshold Variant of the Scheme

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

In [19], 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, 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 an input number to our secret exponent d modulo n^{s+1}. So we can solve our problem by first letting the servers help us compute $E(i,r)^d \bmod n^{s+1}$. Then if we use $g = n+1$ and choose d such that $d = 1 \bmod n^s$ and $d = 0 \bmod \lambda$, the remaining part of the decryption is easy to do without knowledge of d.

We warn the reader that this is only secure for the particular choice of d we have made, for instance, if we had used Paillier's original choice $d = \lambda$, then seeing the value $E(i,r)^d \bmod n^{s+1}$ would allow an adversary to compute λ and break the system completely. However, in our case, the exponentiation result can safely be made public, since it contains no trace of the secret λ.

A more concrete description: Compared to [19] we still have a secret exponent d, but there is no public exponent e, so we will have to do some things slightly differently towards the end of the decryption process. We will assume that there are l decryption servers, and a minimum of $w \leq l/2$ of these are needed to make a correct decryption. We will use as modulus n a product of safe primes, i.e., $n = pq$, where $p, q, p' = (p-1)/2, q' = (q-1)/2$ are primes.

We will need as a subroutine a zero-knowledge proof that for given values $u, \tilde{u}, v, \tilde{v} \in Z^*_{n^{s+1}}$, it holds that $log_u(\tilde{u}) = log_v(\tilde{v})$. Here, it is guaranteed that all values are in the group of squares modulo $Z^*_{n^{s+1}}$, and that v generates the entire group of squares. Note that this group is always cyclic of order $n^s p' q'$, since n is a safe prime product.

A protocol for this can be easily derived from the corresponding one in [19], and works as follows:

Protocol for equality of discrete logs.
Input: $u, \tilde{u}, v, \tilde{v} \in Z^*_{n^{s+1}}$. Private input for P: y such that $y = log_u(\tilde{u}) = log_v(\tilde{v})$ (in our application, the length of y will be at most $(s+1)k$ bits, where k is the modulus length).

1. P chooses a number r at random of length $(s+2)k + t$ bits and sends $a = u^r \bmod n^{s+1}, b = v^r \bmod n^{s+1}$ to the verifier V. Here, t is a (secondary) security parameter.

2. V chooses a random challenge e of length t bits.
3. P sends to V the number $z = r + ey$ and V checks that $u^z = a\tilde{u}^e \bmod n^{s+1}, v^z = b\tilde{v}^e \bmod n^{s+1}$

This protocol can be made non-interactive using the Fiat-Shamir heuristic and a hash function H: the prover computes a, b as above, sets $e = H(a, b, u, \tilde{u})$, computes the reply z as above and defines the proof to be (e, z). To verify such a proof, one checks that $e = H(u^z\tilde{u}^{-e}, v^z\tilde{v}^{-e}, u, \tilde{u})$. Note that we do not need to include $v, \tilde{v}$ in the input to H because in our application, they are fixed and chosen by an honest dealer. Assuming the random oracle model, i.e., replacing H by a random function, one can show soundness and zero-knowledge of this protocol. This is done in exactly the same way as in [19] since, like Shoup, we are working in a cyclic group with only large prime factors in its order. We leave the details to the reader.

Key generation
Key generation starts out as in [19]: 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 $m = p'q'$. We decide on some $s > 0$, thus the plaintext space will be Z_{n^s}. We pick d to satisfy $d = 0 \bmod m$ and $d = 1 \bmod n^s$. Now we make the polynomial $f(X) = \sum_{i=0}^{w-1} a_i X^i \bmod n^s m$, by picking a_i (for $0 < i < w$) as random values from $\{0, \cdots, n^s * m - 1\}$ 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 (n, s). For verification of the actions of the decryption servers, we need the following fixed public values: v, generating the cyclic group of squares in $Z^*_{n^{s+1}}$ and for each decryption server a verification key $v_i = v^{\Delta s_i} \bmod n^{s+1}$, where $\Delta = l!$.

Encryption
To encrypt a message M, a random $r \in Z^*_n$ is picked and the cipher text is computed as $c = (n+1)^M r^{n^s} \bmod n^{s+1}$. As seen in the previous schemes a generator h can be chosen to improve efficiency. Since this only affects the encryption it will not affect the security of the threshold decryption scheme.

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 zero-knowledge proof as described above 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.

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

$$c' = \prod_{i \in S} c_i^{2\lambda_{0,i}^S} \bmod n^{s+1} \qquad \text{where } \lambda_{0,i}^S = \Delta \prod_{i' \in S \setminus i} \frac{-i}{i - i'} \in Z$$

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 $4\Delta^2 d = 4\Delta^2 \bmod n^s$, we can conclude that $c' = (1+n)^{4\Delta^2 M} \bmod n^{s+1}$, where M is the desired plaintext, so this means we can compute M by applying the algorithm from Theorem 1 and multiplying the result by $(4\Delta^2)^{-1} \bmod n^s$.

Compared to the scheme proposed in [12], there are some technical differences, apart from the fact that [12] only works for the original Paillier version modulo n^2: in [12], 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 particular, ideas from [10] can be used to give a reasonably efficient solution.

Note that the key generation phase requires that a value of the parameter s is fixed. This means that the system will be able to handle messages encrypted modulo $n^{s'+1}$, for any $s' \leq s$, simply because the exponent d satisfies $d = 1 \bmod n^{s'}$, for any $s' \leq s$. But it will not work if $s' > s$. If a completely general decryption procedure is needed, this can be done as well: If we assume that λ is secret-shared in the key set-up phase, the servers can compute a suitable d by running a secure protocol that first inverts λ modulo n^s to get some x as result, and then computes the product $d = x\lambda$ (over the integers). This does not require generic multiparty computation techniques, but can be done quite efficiently using techniques from [20]. Note that, while this does require communication between servers, it is not needed for every decryption, but only once for every value of s that is used.

We can now show in the random oracle model that this threshold version is as secure as a centralized scheme where one trusted player does the decryption[3], in particular the threshold version is secure relative to the same complexity assumption as the basic scheme. This can be done in a model where a static adversary corrupts up to $w-1$ players from the start. Concretely, we have:

Theorem 4. *Assume the random oracle model and a static adversary that corrupts up to $w-1$ players from the beginning. Then we have: Given any ciphertext, the decryption protocol outputs the correct plaintext, except with negligible probability. Given an oracle that on input a ciphertext returns the corresponding plaintext, the adversary's view of key generation and of the decryption protocol can be efficiently simulated with a statistically indistinguishable distribution.*

[3] In fact the random oracle will be needed only to ensure that the non-interactive proofs of correctness of shares will work. Doing these proofs interactively instead would allow us to dispense with the random oracle

The proof follows very closely the corresponding proof in [19]. So here we only sketch the basic ideas: correctness of the scheme is immediate assuming that the adversary can contribute incorrect values for the c_i's with only negligible probability. This, in turn, is ensured by soundness of the zero-knowledge proofs given for each c_i.

For the simulation, we start from the public key n. Then we can simulate the shares $s_{i_1}, ..., s_{i_{w-1}}$ of the bad players by choosing them as random numbers modulo n^{s+1}. This will be statistically indistinguishable from the real values which are chosen modulo $n^s p'q'$. Since d is fixed by the choice of n, this means that the shares of uncorrupted players and the polynomial f are now fixed as well, in particular we have $f(i_1) = s_{i_1}, ..., f(i_{w-1}) = s_{i_{w-1}}$. But d, f are not easy for the simulator to compute.

However, if we simulate v by choosing it as a ciphertext with known plaintext m_0, i.e., $v = (n+1)^{m_0} r^{2n^s} \bmod n^{s+1}$, we can also compute what $v^{f(0)}$ would be, namely $v^{f(0)} = v^d \bmod n^{s+1} = (1+n)^{m_0} \bmod n^{s+1}$. Let S be the set $0, i_1, ..., i_{w-1}$ of w indices, and let

$$\lambda^S_{j,i} = \Delta \prod_{i' \in S \setminus i} \frac{j - i}{i - i'}$$

be the Lagrange coefficients for interpolating the value of a polynomial in point j (times Δ) from its values in points in S. Then we can compute correct values of v_j for uncorrupted players as

$$v_j = \prod_{i \in S} (v^{f(i)})^{\lambda^S_{j,i}}.$$

When we get a ciphertext c as input, we ask the oracle for the plaintext m. This allows us to compute $c^d = (1+n)^m \bmod n^{s-1}$. Again this means we can interpolate and compute the contributions c_i from the uncorrupted players. Finally, the zero-knowledge property is invoked to simulate the proofs that these c_i are correct.

5.2 Some Auxiliary Protocols

Suppose a prover P presents a skeptical verifier V with a ciphertext c and claims that it encodes plaintext i, or more precisely that he knows r such that $c = E(i, r)$. A trivial way to convince V would be to reveal also the random choice r, then V can verify himself that $c = E(i, r) = (1+n)^i r^{n^s} \bmod n^{s+1}$. However, for use in the following, we need a solution where no extra useful information is revealed.

It is easy to see that this is equivalent to convincing V that $c(1+n)^{-i} \bmod n^{s+1}$ is an encryption of 0, or equivalently that it is an n^s'th power. So we now propose a protocol for this purpose which is a simple generalization of the one from [15].

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 [5]), which in particular implies that they satisfy standard knowledge soundness.

Protocol for n^s'th powers
Input: n, u
Private Input for P: $v \in Z_n^*$, such that $u = E(0, v)$.

1. P chooses r at random in Z_n^* and sends $a = E(0, r)$ to V
2. V chooses e, a random t bit number, and sends e to P.
3. P sends $z = rv^e \bmod n$ to V. V checks that u, a, z are prime to n and that $E(0, z) = au^e \bmod n^{s+1}$, and accepts if and only if this is the case.

It is now simple to show

Lemma 3. *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 v such that $u = E(0, v)$, provided 2^t is smaller than the smallest prime factor of n.*

Proof. For completeness, we just plug into the equation that V checks, by Lemma 2 we get $au^e = E(0, r)E(0, v)^e = E(0, rv^e \bmod n) = E(0, z) \bmod n^{s+1}$.

For honest verifier simulation, the simulator chooses a random $z \in Z_n^*$, a random e, sets $a = E(0, z)u^{-e} \bmod n^{s+1}$ 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 $E(0, z) = au^e \bmod n^{s+1}$ and $E(0, z') = au^{e'} \bmod n^{s+1}$, so we get

$$E(0, z/z' \bmod n) = u^{e-e'} \bmod n^{s+1}$$

Since $e - e'$ is prime to n by the assumption on 2^t, choose α, β such that $\alpha n^s + \beta(e - e') = 1$. Let $\bar{u} = u \bmod n$ and set $v = \bar{u}^\alpha (z/z')^\beta \bmod n$. Notice that $u^{n^s} \bmod n^{s+1} = E(0, u \bmod n) = E(0, \bar{u})$. We then get

$$E(0, v) = E(0, \bar{u})^\alpha E(0, z/z')^\beta = u^{\alpha n^s} u^{\beta(e-e')} = u \bmod n^{s+1}$$

so that v is indeed the desired n^s'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 [5], 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 $u_1 = C/g^{i_1} \bmod n^{s+1}, u_2 = C/g^{i_2} \bmod n^{s+1}$, and the prover shows that either u_1 or u_2 encrypt 0 and also proves knowledge of one of the corresponding n^s'th roots. This can be done using the following protocol, where we assume without loss of generality that the prover knows v_1 such that $u_1 = E(0, v_1)$, and where M denotes the honest-verifier simulator for the n^s-power protocol above:

Protocol 1-out-of-2 n^s'th power
Input: n, u_1, u_2
Private Input for P: v_1, such that $u_1 = E(0, v_1)$

1. P chooses r_1 at random in Z_n^*. He invokes M on input n, u_2 to get a conversation a_2, e_2, z_2. He sends $a_1 = E(0, r_1), a_2$ to V
2. V chooses s, a random t bit number, and sends s to P.
3. P computes $e_1 = s - e_2 \bmod 2^t$ and $z_1 = r_1 v_1^{e_1} \bmod n$. He then sends e_1, z_1, e_2, z_2 to V.
4. V checks that $s = e_1 + e_2 \bmod 2^t$, $E(0, z_1) = a_1 u_1^{e_1} \bmod n^{s+1}$, $E(0, z_2) = a_2 u_2^{e_2} \bmod n^{s+1}$, and $u_1, u_2, a_1, a_2, z_1, z_2$ are relatively prime to n. He accepts if and only if this is the case.

The proof techniques from [5] and Lemma 3 immediately imply

Lemma 4. *Protocol* 1-out-of-2 n^s'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 v, such that either $u_1 = E(0, v)$ or $u_2 = E(0, v)$, 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 $ab = c \bmod n^s$. For this, we propose a protocol inspired by a similar construction found in [6].

Protocol Multiplication-mod-n^s
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 random values $d \in Z_{n^s}$, $r_d, r_{db} \in Z_n^*$ 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 $e_a^e e_d = E(ea + d, r_a^e r_d \bmod n)$ by sending $f = ea + d \bmod n^s$ and $z_1 = r_a^e r_d \bmod n$. Finally, P opens the encryption $e_b^f (e_{db} e_c^e)^{-1} = E(0, r_b^f (r_{db} r_c^e)^{-1} \bmod n)$ by sending $z_2 = r_b^f (r_{db} r_c^e)^{-1} \bmod n$.
4. V verifies that the openings of encryptions in the previous step were correct, that all values sent by P are relatively prime to n, and accepts if and only if this was the case.

Lemma 5. *Protocol* Multiplication-mod-n^s *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* $ab = c \bmod n^s$, *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 $e_a^e e_d = E(f, z_1) \bmod n^{s+1}$ and $e_b^f (e_{db} e_c^e)^{-1} = E(0, z_2) \bmod n^{s+1}$. 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^{s+1}, 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 $f = ea + d \bmod n^s$, $f' = e'a + d \bmod n^s$, $fb - x - ec = 0 = f'b - x - e'c \bmod n^s$. Putting this together, we obtain $(f - f')b = (e - e')c \bmod n^s$ or $(e - e')ab = (e - e')c \bmod n^s$. Now, since $(e - e')$ is invertible modulo n^s by assumption on 2^t, we can conclude that $c = ab \mod n^s$ (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.

Furthermore, although the protocols here have been phrased so that they can be used to prove statements on values encrypted in CS_s, they can also be directly used in the same way for values encrypted under the more efficient variant $\widetilde{CS}_s$. This follows from the fact that if for a given $u \in Z^*_{n^{s+1}}$, you know $i, \tilde{v}$ such that $u = \tilde{E}(i, \tilde{v})$, we have that $u = E(i, h^{\tilde{v}} \bmod n)$, in other words you can efficiently compute v such that $u = E(i, v)$. Thus a prover can use u in any of the above protocols pretending it was encrypted using CS_s. Note that this applies to both ciphertexts that are input to the protocols, and those that are generated by the prover during executions.

6 Efficient Electronic Voting

In [8], 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 can all be implemented in a secure way for instance 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. Also, we will assume throughout that an instance of the 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 $n^s > M^L$, which can always be made true by choosing s or n large enough.

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 $e = h(a, S, ID(P))$ 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 $Proof_P(S) = (e, z)$. The inclusion of $ID(P)$ in the input to h is done in order to prevent vote duplication. To check such a proof, note that all the auxiliary protocols are such that from S, z, c one can easily compute what a should have been, had the proof been correct. For instance, for the protocol for n^s powers, the statement consists of a single number u modulo n^{s+1}, and the verifier checks that $z^{n^s} = au^e \bmod n^{s+1}$, so we have $a = z^{n^s} u^{-e} \bmod n^{s+1}$. Once a is computed, one checks that $e = h(a, S, ID(P))$.

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 be 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 $E_i/(1+n)$ is an encryption of 0)
based on the 1-out-of-2 n^s'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 $E := E \cdot E_i \bmod n^{s+1}$. 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 $E = \prod_i E(v_i, r_i) = E(\sum_i v_i \bmod n^s, \prod_i r_i \bmod n^{s+1})$. So the decryption result is $\sum_i v_i \bmod n^s$ which is $\sum_i v_i$ since $n^s > M$.

Security of this protocol (in the random oracle model) can be proved based on the security results we have shown for the sub-protocols used, and based on semantic security of Paillier's encryption scheme. Since the voting schemes in this paper play the role of example applications of our crypto system and auxiliary protocols we do not give a formal proof here. However, in [14], Groth presents a full proof of security for our voting scheme according to the definition of Canetti.

There are several ways to generalize 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 the following form (where $j = 1, .., L$):

$$\begin{aligned} E_{ij} =& E(v_{ij}, r_{ij}), \\ & Proof_{V_i}(E_{ij} \text{ or } E_{ij}/(1+n) \text{ is an encryption of } 0) \end{aligned}$$

To prove that he voted for exactly t candidates, he also writes to B the number $\prod_j r_{ij} \bmod n$. 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.

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 $k = 1000, M = 64000, L = 64$, $s = 1$ and we use challenges of 80 bits in the proofs. Then a vote in the above system has size about 32 Kbyte.

We note that this easily generalizes 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.

A more efficient method for large t is to add only 1 dummy candidate who is to receive all unused votes. Each voter must still prove that the product of all his encryptions decrypts to t. So it is sufficient to prove in addition that the number of votes on the dummy candidate is small enough in order that a reduction modulo n^s cannot take place when the votes of this voter are added. This can be done by taking the bit string representing the number of votes on the dummy candidate: $b_0...b_l$ where $2^l \leq t < 2^{l+1}$. The voter then makes encryptions $e_{ij} = E(b_j 2^j, r_{ij})$ for all $0 \leq j \leq l$ and makes a proof for each of these:

$$Proof_{V_i}(e_{ij} \text{ or } e_{ij}/(1+n)^{2^j} \text{ is an encryption of } 0)$$

The votes for the dummy candidate can then be calculated as $E_{iL} = \prod_{i=0}^{l} e_i$. Then it is verified as above that $\prod_{i=0}^{L} E_{ij}$ is the encryption of t. This only uses $L+1$ blocks and $L + \log t$ proofs.

6.1 A variant with smaller vote size

If the parameters are such that $L \log_2 M < (k-1) \cdot s$ and $t = 1$, then we can do significantly better than above. These conditions will be satisfied in many realistic situations, such as for instance in the numeric example above.

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 $a = \sum_{j=0}^{L} a_j M^j \bmod n^s$, where a_j is the number of votes cast for candidate j. Since $L \log_2 M < (k-1) \cdot s$ so that $M^L < n^s$, 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 $0 \leq j < L$, and prove it was correctly formed. We do this in the following two subsections.

We note that this idea generalizes to $t > 1$, at some loss of efficiency: we simply allow each voter to cast t votes, each of the form just described. If we want to prevent voters from voting for the same candidate t times, we can use the homomorphic property to compute encryptions of all pairwise differences of votes, and the voter must prove that these are all non-zero. To show that m is non-zero, given the encryption $E(m, r)$, the voter provides an encryption $E(m^{-1} \bmod n^s, r')$ and uses the *multiplication-mod-n^s* protocol to prove that the product of the two plaintexts is 1.

The case of $L = 2^{l+1}$ For simplicity, we will first describe how to prove correctness of a vote in the case where L is of the form $L = 2^{l+1}$ for some l, and treat the general case below. 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 $Proof_P(e_i/(1+n)$ or $e_i/(1+n)^{M^{2^i}}$ is an encryption of 0).
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
$$Proof_P(\text{Plaintexts corr. to } f_{i-1}, e_i, f_i \text{ satisfy} \quad F_{i-1} \cdot (M^{2^i})^{b_i} = F_i \bmod n^s \ ),$$
based on the multiplication-mod-n^s protocol. The encryption f_l is the desired encryption.

It is straightforward to verity from the e_i, f_i and all the proofs computed that f_l is an encryption of a number of form M^j. Furthermore, simply because there are $l + 1$ encryptions $e_0, .., e_l$ each determining one bit of j, it is clear that $0 \leq j < 2^{l+1} = L$.

It is straightforward to see that a vote in this system will have length $O(k \log L)$ bits (still assuming, of course, that $L \log_2 M \leq (k-1) \cdot s$).

With parameter values as in the numeric example before, a vote will have size about 7 Kbyte, a factor of almost 5 better than the previous system. Moreover, we need only 1 decryption operation as opposed to L before.

The case of general L If L is not of the nice form we assumed above, we may attempt to adapt the above solution as follows: first define l by: 2^{l+1} is the smallest 2-power with $2^{l+1} > L$, and then run the above protocol with no further changes. There are two drawbacks to this idea: first, it allows voters to vote for non-existing candidates, namely j's for which $L \leq j < 2^{l+1}$, and second this also implies that we must have $2^{l+1} \log_2 M \leq (k-1) \cdot s$, otherwise we may get overflow when votes are added, and the result will be incorrect. If we could prevent voters from voting for non-existing candidates, we would only need $L \log_2 M \leq (k-1) \cdot s$, so this simple-minded solution may force us to have a block length larger than what is strictly necessary, in the worst case almost twice as large.

One way to get around this is to add an extra step to the verification of a vote where, given the encryptions $e_0, ..., e_l$ determining the bits of j, the voter proves in zero-knowledge that $j < L$.

To this end, first recall that we defined $j = b_0 2^0 + b_1 2^1 + ... + b_l 2^l$, and that for each encryption e_i that is provided, it is shown that it encrypts $M^{b_i 2^i}$. Define $\beta_i = (M^{2^i} - 1)^{-1} \bmod n^s$. It is now easy to see that

$$\begin{aligned} e'_i &= ((e_i(1+n)^{-1})^{\beta_i} \bmod n^{s+1}) \bmod n^2 \\ &= (e_i(1+n)^{-1})^{\beta_i \bmod n} \bmod n^2 \end{aligned}$$

is an encryption of b_i in CS_1, and furthermore a verifier can compute this value without interaction, from already public information. Going to CS_1 means that the complexity of the protocol to follow becomes independent of s. From this point there are several ways to proceed, we sketch one simple option here:

Let L be represented by bits $B_0, ..., B_l$. We can now exploit the following fact:

$$j < L \text{ iff } \exists\, i, \text{ such that} B_l = b_l, \cdots, B_{i+1} = b_{i+1}, B_i = 1, b_i = 0$$

Notice that $d_i = ((2B_i - 1)(2b_i - 1) + 1)/2$ is a binary value that is 1 if $B_i = b_i$ and 0 otherwise. Since B_i is public, the verifier can compute an encryption of d_i from e'_i without interaction. Clearly, the product $D_i = d_l \cdots d_{i+1} B_i (1 - b_i)$ is 1 precisely if i is an index confirming that $L > j$. It is also easy to see that by providing encryptions of the values $(d_l d_{l-1})$, $(d_l d_{l-1} d_{l-2}), ..., (d_l \cdots d_1)$ and of the D_i's, the prover can show that the encryptions of the D_i's contain correct values, using $2l$ multiplication proofs. Finally, the prover needs to show that one of the $D_i = 1$ for some i. This can be done by a trivial generalization of the one-of-two protocol we showed earlier. In total, this solution will have complexity $O(k \log L)$ bits (assuming that $L \log_2 M \leq (k-1) \cdot s$). This is asymptotically the same as before, but with a larger constant. We note that Lipmaa et al. in [17] have recently proposed a conceptually simpler solution for general L which is more efficient than ours by a constant factor.

References

1. *Algorithmic Number Theory, Volume I: Efficient Algorithms.* Foundations of Computing Series. The MIT Press, Cambridge, Massachusetts; London England, 1996.

2. L. Blum, M. Blum, and M. Shub: *A simple secure unpredictable pseudo-random number generator*, SIAM Journal on Computing, 15(2): pp. 364-383, May 1986.
3. O. Baudron, P.-A. Fouque, D. Pointcheval, G. Poupard and J. Stern: *Practical Multi-Candidate Election Scheme*, Proceedings of PODC 2001.
4. R. Cramer, I. Damgård and J. Nielsen: *Multiparty Computation from Threshold Homomorphic Encryption*, Proceedings of EuroCrypt 2001, Springer Verlag LNCS series 2045, pp.280-300.
5. R. Cramer, I. Damgård and B. Schoenmakers: *Proofs of partial knowledge*, Proceedings of Crypto 94, Springer Verlag LNCS series 839, pp. 174-187.
6. R. Cramer, S. Dziembowski, I. Damgård, M. Hirt and T. Rabin: *Efficient Multiparty Computations Secure against an Adaptive Adversary*, Proceedings of EuroCrypt 99, Springer Verlag LNCS series 1592, pp. 311-326.
7. D. Catalano, R. Gennaro and N. Howgrave-Graham: *The bit security and Paillier's encryption scheme and its applications*, Proceedings of EuroCrypt 2001, Springer Verlag LNCS series 2045, pp. 229-243.
8. R. Cramer, R. Gennaro and B. Schoenmakers: *A Secure and Optimally Efficient Multi-Authority Election Scheme*, Proceedings of EuroCrypt 97, Springer Verlag LNCS series 1233, pp. 103-118.
9. I. Damgård and M. Jurik: *A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System*, Proceedings of Public Key Cryptography 2001, Springer Verlag LNCS series 1992, pp. 119-136.
10. I. Damgård and M. Koprowski: *Practical Threshold RSA Signatures Without a Trusted Dealer*, Proceedings of EuroCrypt 2001, Springer Verlag LNCS series 2045, pp. 152-165.
11. Y. Frankel, P. MacKenzie and M. Yung: *Robust Efficient Distributed RSA-key Generation*, proceedings of STOC 98, pp. 663-672.
12. P.-A. Fouque, G. Poupard and J. Stern: *Sharing Decryption in the Context of Voting or Lotteries*, Proceedings of Financial Crypto 2000.
13. O. Goldreich and V. Rosen: *On the security of modular exponentiation with application to the construction of pseudorandom generators*, Cryptology ePrint Archive, record 2000/064, `http://eprint.iacr.org/`, December 2000.
14. J. Groth: *Extracting Witnesses From Proofs of Knowledge in the Random Oracle Model* , Manuscript, December 2001, Eprint archive report nr. 2002/002.
15. L. Guillou and J.-J. Quisquater: *A Practical Zero-Knowledge Protocol fitted to Security Microprocessor Minimizing both Transmission and Memory*, Proceedings of EuroCrypt 88, Springer Verlag LNCS series 330, pp. 123-128.
16. M. Hirt and K. Sako: *Efficient Receipt-Free Voting based on Homomorphic Encryption*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS series 1807, pp. 539-556.
17. H. Lipmaa, N. Asokan and V. Niemi: *Secure Vickrey Auctions without Threshold Trust*, IACR Eprint archive, 2001/95.
18. P. Paillier: *Public-Key Cryptosystems based on Composite Degree Residue Classes*, Proceedings of EuroCrypt 99, Springer Verlag LNCS series 1592, pp. 223-238.
19. V. Shoup: *Practical Threshold Signatures*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS series 1807, pp. 207-220.
20. J. Bar-Ilan and D. Beaver: *Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds*, Proceedings of the ACM Symposium on Principles of Distributed Computation, 1989, pp. 201-209.