A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System

Ivan Damgård and Mads Jurik

University of Aarhus, **BRICS***

Abstract. We propose a generalisation 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 loosing the homomorphic property. We show that the generalisation is as secure as Paillier's original system.

We construct a threshold variant of the generalised 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 optimised such that for a certain range of parameter values, a ballot has size only $O(\log L)$ bits.

1 Introduction

In [9], 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 generalisation 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 loosing the homomorphic property. The generalisation also allows reducing the expansion factor from 2 for Paillier's original system to almost 1. We prove that the generalisation is as secure as Paillier's original scheme.

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

We propose a threshold variant of the generalised 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 centralised 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 [4]. 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 [9] that since his encryption scheme is homomorphic, it may be applicable to electronic voting. In order to apply it in the framework of [4], 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 [4]. However, the work needed to produce the result is reduced dramatically, as we now explain. With the El Gamal encryption used in [4], 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 [4] 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 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 $L > 2$.

2 Related Work

In work independent from, but earlier than ours, Fouque, Poupard and Stern [6] proposed the first threshold version of Paillier's original scheme. Like our threshold scheme, [6] uses an adaptation of Shoup's threshold RSA scheme [10], but beyond this the techniques are somewhat different, in particular because we construct a threshold version for our generalised crypto system (and not only Paillier's original scheme). In [6] 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 [1] 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 towards the system architectural aspects of a large scale election, and less towards optimisation of the building blocks. To compare to their scheme, we first note that there the modulus length k must be chosen such that $k > L \log M$. The scheme produces ballots of size $O(k \cdot L)$. An estimate with explicit constants is given in [1] in which the dominating term in our notation is $9kL$.

Because our voting scheme uses the generalised Paillier crypto system, k can be chosen freely, and the voting scheme can still accommodate any values of L, M. If we choose k as in [1], i.e. $k > L \log M$, then the ballots we produce have size $O(k \cdot \log L)$. Working out the concrete constants involved, one finds that our complexity is dominated by the term $11k \log L$. So for large scale elections we have gained a significant factor in complexity compared to [1].

In [8], 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 [4]. It can also be applied to our scheme, with the same efficiency gain as in the non-receipt free case.

[1] All complexities given here assume that the length of challenges for the zero-knowledge proofs is at most k. Also, strictly speaking, this complexity only holds if $k > \log M$, however, since $k \geq 1000$ is needed for security anyway, this will always be satisfied in practice

3 A Generalisation 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 [9] as a special case by setting $s = 1$.

We start from the observation that if $n = pq$, p, q odd primes, then $Z^*_{n^{s+1}}$ 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, which follows directly from elementary number theory. 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 $s < p, q$, 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 the order of H is relatively prime to n^s this 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 final technical observation we need is that it is easy to compute i from $(1 + n)^i \bmod n^{s+1}$. We now show how to do this. 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
  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
```

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

Key Generation On input the security parameter k, choose an 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 $r \in Z^*_{n^{s+1}}$, 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 $c = E(v,r)$, we get

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

Now apply the above algorithm 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 $i + i' \bmod n^s$.

The security of the system is based on the following assumption, introduced by Paillier in [9] 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 cryptosystem gets the public key pk generated from secuity parameter k as input and outputs a message m. Then A is given an encryption under pk of either m or a message*

[2] strictly speaking, we also need that $s < p, q$, but this is insignificant since s is a constant

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 cryptosystem is* semantically secure *if for any probabilistic polynomial time adversary A, $Adv(A,k)$ is negligible in k.*

In [9], Paillier showed that semantic security of his cryptosystem (which is the same as our CS_1) is equivalent to DCRA. This equivalence 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, $cg^{-i} \bmod n^2$ is either random in $Z^*_{n^2}$ or a random n'th power. In particular one may choose $g = n+1$ always without degrading security. We do this in the following for simplicity, so that a public key consists only of the modulus n. We now show that in fact security of CS_s is equivalent to DCRA:

Theorem 1. *For any s, the cryptosystem CS_s is semantically secure if and only if the DCRA assumption is true.*

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 show by induction on s that security of CS_s follows from DCRA. For $s = 1$, this is exact.ly Paillier's result. So take any $s > 1$ and assume that CS_t for any $t < s$ is secure.

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 contradiction that CS_s is insecure, thus there is an adversary A, such that for infinitely many k, $Adv(A,k) \geq 1/f(k)$ for some polynomial $f()$. Take such a k. Without loss of generality, assume 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 $D_n(*, m_{s-1}, ..., m_1)$. Let $q(A,k)$ be the probability that A now outputs 1. Of course, we must have

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

for infinitely many k.

In the first case in $(*)$, we can make a successful adversary against CS_1, as follows: we get the public key n, show it to A, get $(m_s, ..., m_1)$, and return m_s as output. We will get a ciphertext c that either encrypts m_s in CS_1, or is a random ciphertext, i.e., a random element from $Z^*_{n^2}$. 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

or a random element in its least significant position. Hence $c^{n^{s-1}} \bmod n^{s+1}$ is an encryption of $(m_s, 0, ..., 0)$ or $(*, 0, ..., 0)$. We then make a random encryption d of $(0, m_{s-1}, ..., m_1)$, give $c^{n^{s-1}} d \bmod n^{s+1}$ to A and return the bit A outputs. Now, if c encrypts m_s, we have shown to A a sample of $D_n(m_s, ..., m_1)$, and otherwise a sample of $D_n(*, m_{s-1}, ..., m_1)$. So by assumption on A, this breaks CS_1 with an advantage of $1/2f(k)$, and so contradicts the induction assumption.

In the second case of $(*)$, we can make an adversary against CS_{s-1}, as follows: we get the public key n, show it to A, and get a message $(m_s, ..., m_1)$. We output $(m_{s-1}, ..., m_1)$ and get back a ciphertext c that encrypts in CS_{s-1} either $(m_{s-1}, ..., m_1)$ 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 $(m_{s-1}, ..., m_1)$ or random elements in the least significant $s-1$ positions - and something unknown in the top position. 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 $(m_{s-1}, ..., m_1)$, we have shown A a sample from $D_n(*, m_{s-1}, ..., m_1)$, and otherwise a sample from $D_n(*, ..., *)$. So by asumption on A, this breaks CS_{s-1} with an advantage of $1/2f(k)$ and again contradicts the induction assumption.

3.1 Adjusting the Block length

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 mentioned, we may choose $g = n + 1$ always without loosing security, and the public key may then consist only of the modulus n. This means that we can let the receiver decide on s when he encrypts a message. More concretely, the system will work as follows:

Key Generation Choose an 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 $i \in Z_{n^s}$, choose a random $r \in Z^*_{n^{s+1}}$, and let the ciphertext be $E(i, r) = (1+n)^i r^{n^s} \bmod n^{s+1}$.

decryption Given a ciphertext c, first compute, by the Chinese Remainder Theorem d, such that $d = 1 \bmod n^s$ and $d = 0 \bmod \lambda$ (note that the length of the ciphertext allows to decide on the right value of s, except with negligible probability). Then compute $c^d \bmod n^{s+1}$. Clearly, if $c = E(i, r)$, we get

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

Now apply the above algorithm to compute $i \bmod n^s$.

4 Some Building Blocks

4.1 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 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 [10], 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 [10] we still have a secret exponent d, but there is no public exponent e, so we will have to do some things slightly differently. We will assume that there are l decryption servers, and a minimum of $k < n/2$ of these are needed to make a correct decryption.

Key generation
Key generation starts out as in [10]: 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 $f(X) = \sum_{i=0}^{k-1} a_i X^i \bmod n^s m$, by picking a_i (for $0 < i < k$) 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 $1 \le i \le l$ and the public key will be 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 $Z^*_{n^{s+1}}$ 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 $r \in Z^*_{n^{s+1}}$ is picked and the cipher text is computed as $c = g^M r^{n^s} \bmod n^{s+1}$.

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 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 [3]

[3] A non interactive zero-knowledge proof for this using the Fiat-Shamir heuristic is easy to derive from the corresponding one in [10]

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

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

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 Section 3 and multiplying the result by $(4\Delta^2)^{-1} \bmod n^s$.

Compared to the scheme proposed in [6], there are some technical differences, apart from the fact that [6] only works for the original Paillier version modulo n^2: in [6], 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 multi-party computation techniques it is also possible to do the key generation without a trusted party.

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 multi-party computation techniques, but can be done quite efficiently using techniques from [5]. 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 centralised scheme where one trusted player does the decryption[4], 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 $k-1$ players from the start. Concretely, we have:

Theorem 2. *Assume the random oracle model and a static adversary that corrupts up to $k-1$ players from the beginning. Then we have: Given any cipher-*

[4] 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

text, 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 the decryption protocol can be efficiently simulated with a statistically indistinguishable distribution.

The full proof will be included in the final version of this paper. Here we only give the basic ideas: correctness of the scheme is immediate assuming that the adversary can contribute bad 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 $s_{i_1}, ..., s_{i_{k-1}}$ of the bad players by choosing them as random numbers in an appropriate interval. 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, but are not easy for the simulator to compute.

However, if we choose v as a ciphertext with known plaintext m_0, we can also compute what $v^{f(0)}$ would be, namely $v^{f(0)} = v^d \bmod n^{s+1} = (1+n)^{m_0} \bmod n^{s+1}$. Then by doing Lagrange interpolation "in the exponent" as in [10], we can compute correct values of $v_i = v^{\Delta s_i}$ for the uncorrupted players. 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.

4.2 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 $cg^{-i} \bmod n^{s+1}$ is an n^s'th power. So we now propose a protocol for this which is a simple generalisation of the one from [7]. 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 [2]), which in particular implies that they satisfy standard knowledge soundness.

Protocol for n^s'th powers
Input: n, u
Private Input for P: v, such that $u = v^{n^s} \bmod n^{s+1}$

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

It is now simple to show

Lemma 2. *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^s'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 $z \in Z^*_{n^{s+1}}$, a random e, sets $a = z^{n^s} u^{-e} \bmod n^{s+1}$ 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 $z^{n^s} = au^e \bmod n^{s+1}$ and ${z'}^{n^s} = au^{e'} \bmod n^{s+1}$, so we get

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

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

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

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 multi-party 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 [2], 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 is an n^s'th power. This can be done using the following protocol, where we assume without loss of generality that the prover knows an n^s'th root u_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 $u_1 = v_1^{n^s} \bmod n^{s+1}$

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

The proof techniques from [2] and Lemma 2 immediately imply

Lemma 3. *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 an n^s'th root of u_1, and an n^s'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 $ab = c \bmod n^s$. For this, we propose a protocol inspired by a similar construction found in [3].

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 a random value $d \in Z_{n^s}$ 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 $e_a^e e_d = E(ea + d, r_a^e r_d \bmod n^{s+1})$ by sending $f = ea + d \bmod n^s$ and $z_1 = r_a^e r_d \bmod n^{s+1}$. Finally, P opens the encryption $e_b^f (e_{db} e_c^e)^{-1} = E(0, r_b^f (r_{db} r_c^e)^{-1} \bmod n^{s+1})$ by sending $z_2 = r_b^f (r_{db} r_c^e)^{-1} \bmod n^{s+1}$.
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 4. *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.

5 Efficient Electronic Voting

In [4], 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 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 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 $n^s > M$, 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 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^s'th power modulo $n^{s+1})$

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) 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 ($j = 1, .., L$)

$$\begin{aligned} E_{ij} =& E(v_{ij}, r_{ij}), \\ & Proof_{V_i}(E_{ij} \text{ or } E_{ij}/g \text{ is an } n^s\text{'th power modulo } n^{s+1}) \end{aligned}$$

To prove that he voted for exactly t candidates, he also writes to B the number $\prod_j r_{ij} \bmod n^{s+1}$. 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 $k = 1000, M = 64000, L = 64$, $s = 1$ and we use challenges of 80 bits in the proofs. Then a vote in the above system has size about 50 Kbyte.

If the parameters are such that $L \log_2 M < k \cdot s$ 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.

The basic idea is the following: a vote for candidate j, where $0 \le j < L$ 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 $L \log_2 M < k \cdot s$, 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. 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 $Proof_P(e_i/g \text{ or } e_i/g^{M^{2^i}}$ is an n^s'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

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

   based on the multiplication-mod-n^s 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 $L \log_2 M \leq k \cdot s$).

With parameter values as in the numeric example before, a vote will have size about 8.5 Kbyte, a factor of more than 5 better than the previous system. Moreover, we need only 1 decryption operation as opposed to L before.

6 Efficiency and Implementation Aspects

An implementation of some of the teqniques discussed in this paper can be found at `http://www.brics.dk/~jurik/research.html`.

Key Generation. The primes p and q are made using the usual techniques, so that n will be as difficult as possible to factor. Since there is no difference in choosing a general g and $(n+1)$ as generator, we can just use $(n+1)$ and save some work for finding a suitable g.

Encryption. As mentioned in Paillier we can choose $g = 2$ (provided it satisfies the contraints) to get a speed-up in encryption. But since we can use $(n+1)$ as generator we can make it even more efficient since calculating $(n+1)^i$ is the same as calculating:

$$1 + in + \binom{i}{2} n^2 + ... + \binom{i}{s} n^s \bmod n^{s+1}$$

this means raising $(n+1)$ to i'th power takes about $5s$ multiplications. We can precompute the factors $k!^{-1} n^k \bmod n^{s+1}$ which reduces the number of multiplications to $2s$. We can't get rid of the exponentiation $r^{n^s} \bmod n^{s+1}$, but the random value can be choosen in advance and the exponentiation calculated in advance. If $r^{n^s} \bmod n^{s+1}$ is calculated in advance an encryption will take $2s$ multiplications which is approximately as efficient as RSA for small s.

Decryption. Decryption can be speeded up by calculating the different powers of n, and the $k!^{-1} \bmod n^j$ for $2 \leq k \leq j \leq s$. All this can be calculated $\bmod p^j$ and $\bmod q^j$ instead of $\bmod n^j$ by using

$$L_p(x) = \frac{x-1}{p} \text{ and } L_q(x) = \frac{x-1}{q}$$

instead of the normal L. The decryption algorithm is then executed 2 times, once $\bmod p^j$'s instead of $\bmod n^j$ and with L_p instead of L and once with $\bmod q^j$ and L_q. Then after the 2 parts have been calculated they are combined using Chinese remaindering.

Performance Evaluations. We give here a comparison between the schemes presented in this paper, Paillier's original scheme, RSA with public exponent $2^{16}+1$ and El-Gamal. There are 3 versions of our scheme, namely one without precomputation, one with, and one with $s = 1$ (and no precomputation), since this is equivalent to Paillier's scheme. It is assumed that all numbers has about the same number of 1's and 0's in their binary representation. In figure 1 we compare the different scheme using the same security parameter. It should be noted that it compares the number of multiplications, but these multiplications are made using different modulus size. It should be also be noted that the 2 first columns encrypt sk bits of plaintext instead of k bits in the other columns. The last 2 rows of the table shows the number of bits that are encrypted for each multiplication made. It only makes sense to compare the numbers in these 2 rows if the modulus size is the same and thus the security parameter k is different.

Fig. 1. Comparison with equal security parameter k

Scheme	General Scheme		Scheme			
	No Precomp.	Precomp.	$s=1$	Paillier	RSA	El-Gamal
n/p Size (k)	k	k	k	k	k	k
Modulus Size	$(s+1)k$	$(s+1)k$	$2k$	$2k$	k	k
Plaintext Size	sk	sk	k	k	k	k
Multiplications for Encryption	$\frac{3}{2}sk+5s$	$2s$	$\frac{3}{2}k+5$	$3k+1$	17	$3k+1$
Multiplications for Decryption	$\frac{5}{2}(s+1)k+$ $2s(s+1)$	$\frac{5}{2}(s+1)k+$ $s(s+1)$	$5k+8$	$\frac{3}{2}k$	$3k+3$	$\frac{3}{2}k+1$
Multiplications per bit encrypted	$\approx \frac{3}{2}$	$\frac{2}{k}$	$\frac{3}{2}$	3	$\frac{17}{k}$	3
Multiplications per bit decrypted	$\approx \frac{5}{2}$	$\approx \frac{5}{2}$	5	3	$\frac{3}{2}$	$\frac{3}{2}$

References

1. Baudron, Fouque, Pointcheval, Poupard and Stern: *Practical Multi-Candidate Election Scheme*, manuscript, May 2000.
2. Cramer, Damgård and Schoenmakers: *Proofs of partial knowledge*, Proc. of Crypto 94, Springer Verlag LNCS series nr. 839.
3. 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.
4. 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.
5. Frankel, MacKenzie and Yung: *Robust Efficient Distributed RSA-key Generation*, proceedings of STOC 98.
6. P. Fouque, G. Poupard, J. Stern: *Sharing Decryption in the Context of Voting or Lotteries*, Proceedings of Financial Crypto 2000.
7. 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.
8. M.Hirt and K.Sako: *Efficient Receipt-Free Voting based on Homomorphic Encryption*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS series, pp. 539-556.
9. P.Pallier: *Public-Key Cryptosystems based on Composite Degree Residue Classes*, Proceedings of EuroCrypt 99, Springer Verlag LNCS series, pp. 223-238.
10. V.Shoup: *Practical Threshold Signatures*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS series, pp. 207-220.
11. 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.