BRICS RS-03-16 Damgård & Jurik: A Length-Flexible Threshold Cryptosystem with Applications

BRICS

Basic Research in Computer Science

A Length-Flexible Threshold Cryptosystem with Applications

Ivan B. Damgård
Mads J. Jurik

BRICS Report Series **RS-03-16**

ISSN 0909-0878 **March 2003**

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/03/16/`

A Length-Flexible Threshold Cryptosystem with Applications

Ivan Damgård and Mads Jurik

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

Abstract. We propose a public-key cryptosystem which is derived from the Paillier cryptosystem. The scheme inherits the attractive homomorphic properties of Paillier encryption. In addition, we achieve two new properties: First, all users can use the same modulus when generating key pairs, this allows more efficient proofs of relations between different encryptions. Second, we can construct a threshold decryption protocol for our scheme that is length flexible, i.e., it can handle efficiently messages of arbitrary length, even though the public key and the secret key shares held by decryption servers are of fixed size. We show how to apply this cryptosystem to build:
1) a self-tallying election scheme with perfect ballot secrecy. This is a small voting system where the result can be computed from the submitted votes without the need for decryption servers. The votes are kept secret unless the cryptosystem can be broken, regardless of the number of cheating parties. This is in contrast to other known schemes that usually require a number of decryption servers, the majority of which must be honest.
2) a length-flexible mix-net which is universally verifiable, where the size of keys and ciphertexts do not depend on the number of mix servers, and is robust against a corrupt minority. Mix-nets can provide anonymity by shuffling messages to provide a random permutation of input ciphertexts to the output plaintexts such that no one knows which plaintexts relate to which ciphertexts. The mix-net inherits several nice properties from the underlying cryptosystem, thus making it useful for a setting with small messages or high computational power, low-band width and that anyone can verify that the mix have been done correctly.
Keywords: length-flexible, length-invariant, mix-net, group decryption, self-tallying, election, perfect ballot secrecy.

1 Introduction

1.1 Background

In [5], Paillier proposed a public-key cryptosystem which, like RSA, uses computations with a composite modulus, but has some very at-

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

tractive properties making it potentially interesting for applications such as electronic voting and mix-nets: it is additively homomorphic and decryption is efficient for all ciphertexts: there is no need to solve discrete logarithms, as with additive homomorphic schemes derived from El-Gamal encryption.

In [11] the system was generalized to handle arbitrary size messages (with the same modulus) and a threshold decryption protocol was proposed. Unfortunately, this protocol can only handle efficiently messages of length smaller than a threshold set at key generation time, for longer messages a cumbersome multiparty computation protocol is needed.

Another unsatisfactory property is that the secret key is essentially the factorization of the modulus, so different users cannot use the same modulus. This means that natural tasks such as proving in zero-knowledge that two ciphertexts (from different public keys) contain the same message become difficult and seem to require generic solutions.

One possible application of homomorphic encryption is to build mix-nets. These are protocols used to provide anonymity for senders by collecting encrypted messages from several users and have a collection of servers process these, such that the plaintext messages are output in randomly permuted order. A useful property for mix-nets is length-flexibility, which means that the mix-net is able to handle messages of arbitrary size. More precisely, what we mean by this is the following: although all messages submitted to a single run of the mix-net must have the same length in order not to break the anonymity, this common length can be decided freely for each run of the mix-net, without having to change any public-key information. This is especially useful for providing anonymity for e.g. E-mails. One way to achieve length-flexibility is to use hybrid mix-nets. These mix-nets use a public key construction to create keys for a symmetric cipher that is used for encrypting the bulk of the messages.

Two length-flexible hybrid mix-nets have been proposed. Ohkubo and Abe in [6] proposed a scheme in which verification of server behavior relies on a generic method by Desmedt and Kurosawa [7]. This results in a system that is robust when at most the square root of the number of mix servers are corrupt. After this Juels and Jakobsson suggested in [13] that verification can be added by using message authentication codes (MACs), which are appended to the plaintext for each layer of encryption. This allows tolerating more corruptions at the expense of efficiency - for instance, the length of the ciphertexts now depends on the number of mix servers as opposed to [6], and each server has to store more secret material. Although the system is verifiable, it is not universally verifiable, which means that external observers cannot verify that everything was done correctly.

In [4] (with some minor pitfall corrected in [10]), Abe introduced verifiable mix-nets using a network of binary switching gates, which is based on the permutation networks of Waksman [1]. This mix-network is robust with up to half of the mix servers being controlled by an active and malicious adversary. One approach to make this length-flexible would be to exchange El-Gamal with a verifiable length-flexible encryption scheme. The cryptosystem in [11] however does not support efficient and length-flexible threshold decryption.

Another application area for homomorphic encryption is electronic voting. In [17] Kiayias and Yung introduced a new paradigm for electronic voting, namely protocols that are self-tallying, dispute-free and have perfect ballot secrecy (STDFPBS for short). This paradigm is suitable for, e.g. boardroom elections where a (small) group of users want a maximally secure vote without help from external authorities. The main property is perfect ballot secrecy, which means that for *any* coalition of voters (even a majority) the only information they can compute is what follows from the result and their own votes, namely the tally of honest users' votes. This is the best we can hope for, and is the type of privacy that is actually achieved by paper based elections. Self-tallying means that as soon as all votes have been cast, no further interaction is needed to compute the result, it can be efficiently computed by just looking at all ballots, which can be done, even by a (casual) third party. Dispute-freeness means that no disputes between players can arise, because all faults are detected in public.

In [17], it is argued that STDFPBS elections cannot be achieved efficiently by traditional methods. For instance, large scale solutions are typically not of this type because they assume that some set of authorities are available to help with the election. The authorities typically share a secret key that can be reconstructed by a majority. In a small scale scenario we could let each voter play the role of an authority himself, but this would not give perfect ballot secrecy because a corrupt majority would know how *every* single voter voted. If we try to repair this by setting the threshold of the secret sharing scheme to be the total number of voters, then even a single fault will mean that the secret key is lost, and an expensive key generation phase would be needed.

In [17] STDFPBS elections are achieved for a yes/no vote by using constructs based on discrete log modulo a prime. This results in a tallying phase that needs to find a discrete log, which requires $O(\sqrt{u})$ work when there are u voters. It also implies that generalization to multi-way elections either results in larger ballots or much worse complexity for the tallying phase. Given earlier work on electronic voting, it is natural to speculate that this could be solved simply by using Paillier encryption instead. However as noted in [17], this does not work, we would lose some essential properties of the scheme.

1.2 Our Contribution

In this paper, we suggest a new public-key cryptosystem. It is a further development of the scheme from [11], it is as efficient up to a constant factor and inherits the homomorphic property. It is semantically secure based on the Paillier and composite DDH assumptions, or - at a moderate loss of efficiency - based only on the Paillier assumption. It is also related to a Paillier-based scheme presented in [15], but is more efficient and is also length-flexible.

We achieve two new properties. First, our scheme allow several users to use the same modulus. This allows efficient zero-knowledge proofs for relations between ciphertexts created under different public keys. We apply this to construct STDFPBS elections, where the tallying phase reveals the result with a small number of additions, instead of $O(\sqrt{u})$ multiplications as in [17]. This also shows that STDFPBS elections with all the essential properties can be based on Paillier encryption, thus solving a problem left open in [17]. Finally, it implies a natural and efficient generalization to multi-way elections.

Second, we propose a threshold decryption protocol where keys can be set up so that messages of arbitrary length can be handled efficiently with the same (fixed size) keys. In addition, the computational work done by each server does not depend on the message length, only the cost of a final public post-processing is message dependent. We also give efficient zero-knowledge protocols for proving various claims on encrypted values.

We combine these with ideas from [4, 6] to construct a mix-net that has several desirable properties at the same time:

– ***Length-flexible:*** The public key does not limit the size of plaintexts that can be encrypted and mixed efficiently. The length of ciphertexts in a specific mix have to be fixed or anonymity will be compromised.
– ***Length-invariant:*** The lengths of the keys and ciphertexts do not depend on the number of mix servers.
– ***Provably secure:*** The system is provable secure in the random oracle model under the Decisional Composite Residuosity Assumption and composite DDH
– ***Universally verifiable:*** Anyone can verify the correctness of the output from the mix-net.
– ***Strong correctness:*** Messages submitted by malicious users cannot be changed once they have been submitted. This hold even in the case of helping malicious servers.
– ***Order flexible:*** The mix servers do not need to be invoked in a certain order. This improves resilience to temporary server unavailability.

We note that all this is achieved by using public key encryption everywhere, which in the passive adversary case makes it less efficient than the Hybrid mix-nets that uses symmetric key crypto to encrypt the messages.

2 Preliminaries

The notion of semantic security that will be used for the encryption system is:

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

The systems in this paper use a modified version of the Damgård-Jurik generalization [11] of the Paillier cryptosystem [5]. The security of this encryption scheme depends on the Decisional Composite Residuosity Assumption first introduced by Paillier.

Conjecture 1 (The Decisional Composite Residuosity Assumption (DCRA)). Let $\mathcal{A}$ be any probabilistic polynomial time algorithm, and assume $\mathcal{A}$ gets n, x as input. Here $n = pq$ is an admissible RSA modulus of length k bits, and x is either random in $\mathbb{Z}_{n^2}^*$ or it is a random n'th power in $\mathbb{Z}_{n^2}^*$. $\mathcal{A}$ outputs a bit b. Let $p_0(\mathcal{A}, k)$ be the probability that $b = 1$ if x is random in $\mathbb{Z}_{n^2}^*$, and $p_1(A, k)$ the probability that $b = 1$ if x is a random n'th power. Then $|p_0(\mathcal{A}, k) - p_1(\mathcal{A}, k)|$ is negligible in k.

The encryption is extended with some discrete logarithm constructions, so the DDH assumption is also needed in a slightly modified version to capture the group.

*Conjecture 2 (The Decisional Diffie-Hellman (composite-DDH))*Let $\mathcal{A}$ be any probabilistic polynomial time algorithm, and assume $\mathcal{A}$ gets $(n, g, g^a \bmod n, g^b \bmod n, y)$ as input. Here $n = pq$ is an admissible RSA modulus of length k bits, g is a element of Q_n the group of squares in $\mathbb{Z}_n^*$. The values a and b are chosen uniformly random in $\mathbb{Z}_{\phi(n)/4}$ and the value y is either random in Q_n or satisfies $y = g^{ab} \bmod n$. $\mathcal{A}$ outputs a bit b. Let $p_0(\mathcal{A}, k)$ be the probability that $b = 1$ if y is random in Q_n, and $p_1(\mathcal{A}, k)$ the probability that $b = 1$ if $y = g^{ab} \bmod n$. Then $|p_0(\mathcal{A}, k) - p_1(\mathcal{A}, k)|$ is negligible in k.

Note that the number $\phi(n)/4$ is not publicly known, so we cannot choose an exponent r with the uniform distribution over $\mathbb{Z}_{\phi(n)/4}$. Throughout this paper, when an exponent is chosen, it will be from the group $\mathbb{Z}_N$, where N is a sufficiently large value. This is done to get a distribution of the elements g^r that is statistically close to the uniform distribution over the group generated by g. To get a difference in distribution smaller than $1/2^k$ the value $N = 2^{|n|+k}$ can be used. A result by Goldreich and Rosen [8] shows that $g^r \bmod n$ for $r \in \{0, ..., 2^{|n|/2} - 1\}$ is a pseudo random generator. So $N = 2^{|n|/2}$ is another possible choice and will result in elements g^r (for $r \in \mathbb{Z}_N$) that cannot be distinguished from an uniform element from the whole group generated by g.

In the Damgård and Jurik paper [11] an algorithm for calculating the discrete logarithm with respect to the element $(n+1)$ is described. In this paper we will use L_s to denote an application of this function, that satisfies:

$$L_s((n+1)^m \bmod n^{s+1}) = m \bmod n^s$$

computing this function requires work $O(s^4k^2) = O(s^2|n^s|^2)$

3 The Basic Cryptosystem

Key Generation: Choose an RSA modulus $n = pq$ of length k bits, with $p = 2p' + 1$ and $q = 2q' + 1$ where p, q, p', q' are primes. Select an element $g \in \mathrm{Q}_n$, the group of all squares of $\mathbb{Z}_n^*$, and $\alpha \in \mathbb{Z}_\tau$, where $\tau = p'q' = |\mathrm{Q}_n|$. The public key is then (n, g, h) with $h = g^\alpha \bmod n$ and the private key is α.

Encryption: Given a plaintext $m \in \mathbb{Z}^+$, choose an integer $s > 0$ such that $m \in \mathbb{Z}_{n^s}$ and a random $r \in \mathbb{Z}_N$, and let the ciphertext be

$$E_s(m, r) = (g^r \bmod n, (h^r \bmod n)^{n^s}(n+1)^m \bmod n^{s+1})$$

Decryption: Given a ciphertext $c = (G, H) = E_s(m, r)$, s can be deduced from the length of c (or attached to the encryption) and m can be found as

$$\begin{aligned} m &= L_s(H(G^\alpha \bmod n)^{-n^s}) \\ &= L_s((g^{\alpha r} \bmod n)^{n^s}(n+1)^m(g^{r\alpha} \bmod n)^{-n^s}) \\ &= L_s((n+1)^m \bmod n^{s+1}) = m \bmod n^s \end{aligned}$$

Remark 1. The key generation above assumes that one knows τ and hence the factorization when choosing α. However, one can also choose α at random in $\mathbb{Z}_N$, that is, generate $\alpha, h = g^\alpha$ from n, g only. This makes no difference to security, since the h produced will have an indistinguishable distribution, and it allows to have n, g be system constants used by all users.

3.1 Security of the Cryptosystem

Theorem 1 (Semantic Security). *Under conjecture 1 (DCRA) and conjecture 2 (composite-DDH) the cryptosystem is semantically secure with respect to definition 1.*

The proof of this theorem can be seen in appendix A. This cryptosystem may seem to be simply a combination of El-Gamal and Paillier, and hence its security is inherently based on both the above conjectures, but this is in fact not true. A simple modification makes it possible to show semantic security based *only* on conjecture 1, using a technique from [15]. Given the same public key (n, g, h), we can set $g' = g^{n^s} \bmod n^{s+1}$ (so that g' generates the subgroup of $\mathbb{Z}^*_{n^{s+1}}$ of order τ), and $h' = h^{n^s} \bmod n^{s+1}$. Encryption is $E'_s(m, r) = (g'^r \bmod n^{s+1}, h'^r(n+1)^m \bmod n^{s+1})$.

Theorem 2 (Semantic Security, modified system). *Under conjecture 1 (DCRA) the modified cryptosystem is semantically secure with respect to definition 1.*

This was proved for $s = 1$ in [15], and follows in general in essentially the same way: a ciphertext (G, H) is always of form $(G, G^\alpha(n+1)^m \bmod n^{s+1})$. Now, conjecture 1 implies that one cannot distinguish the case where $G \in < g' >$ from the case where it is chosen randomly in $\mathbb{Z}^*_{n^{s+1}}$ (with Jacobi symbol 1 with respect to n). In the latter case, however, one can verify that if α is chosen large enough, the ciphertext contains no Shannon information on the plaintext.

Since our basic system is more efficient, we describe our protocols in the following in terms of the $E_s()$ encryption function, but they can all be modified to use the $E'_s()$ encryption function instead.

3.2 An Efficient Length-Flexible Threshold Cryptosystem

From the basic cryptosystem, a length-flexible threshold cryptosystem can be constructed by using a threshold exponentiation based on Shoups threshold signatures [9].

Key Generation: Choose an RSA modulus $n = pq$ and a $g \in \mathrm{Q}_n$ as above. Pick the secret value $a_0 = \alpha \in \mathbb{Z}_\tau$ and some random coefficients $a_i \in \mathbb{Z}_\tau$ for $1 \le i \le t$, where $t \ge w/2$ is the threshold of the system with w servers. The polynomial $f(x) = \sum_{0 \le i \le t} a_i x^i$

is created and the secret shares are calculated as $\alpha_i = f(i)$. The public value is $h = g^\alpha \bmod n$, and the values for verification are $h_i = g^{\alpha_i} \bmod n$. The public key is (n, g, h), the verification values $(h_1, ..., h_w)$, and the private key of server i is α_i.

Encryption: Given a plaintext $m \in \mathbb{Z}^+$, choose an integer $s > 0$, such that $m \in \mathbb{Z}_{n^s}$, and pick a random $r \in \mathbb{Z}_N$. The ciphertext is then

$$E_s(m, r) = (g^r \bmod n, (h^{4\Delta^2 r} \bmod n)^{n^s}(n+1)^m \bmod n^{s+1})$$

Threshold Decryption: Given a ciphertext $c = (G, H) = E_s(m, r)$ each of the servers release the value:

$$d_i = G^{2\Delta\alpha_i} \bmod n$$

and a proof that $\log_g(h_i) = \log_{G^{4\Delta}}(d_i^2)$. The proof used for this is shown in section 3.4. The d_i values, from the set S of servers with legal proofs, are combined using Lagrange interpolation to create the exponent $4\Delta^2\alpha$:

$$d = \prod d_i^{2\lambda_i^S} = G^{4\Delta^2\alpha} = h^{4\Delta^2 r} \bmod n \quad \text{where } \lambda_i^S = \prod_{j \in S \setminus \{i\}} \Delta \frac{j}{j-i}$$

The reason for the factor Δ is to ensure $\lambda_i^S \in \mathbb{Z}$. Now the $h^{4\Delta^2 r}$ can be removed by calculating

$$H' = Hd^{-n^s} = (n+1)^m \bmod n^{s+1}$$

and the plaintext found as $m = L_s(H' \bmod n^{s+1}) \bmod n^s$.

3.3 A Proof Friendly Variant

The system from the previous section only works as long as legal encryptions are submitted, so we would like a protocol allowing us to prove in zero-knowlegde that a ciphertext is well formed. A standard problem with building an efficient protocol of this type is that we need all group elements used to have only large prime factors in their orders. In our case, this can be ensured by squaring them before we start the proofs, i.e., instead of trying to show that g has some desired property, we prove that g^2 has it. However, as we shall see, this only implies that g or $-g$ has the desired property. To handle this, we define a slightly relaxed cryptosystem:

Key Generation: As above

Encryption: Given a plaintext $m \in \mathbb{Z}^+$, choose an integer $s > 0$, such that $m \in \mathbb{Z}_{n^s}$, and pick a random $r \in \mathbb{Z}_N$ and $b_0, b_1 \in \{0, 1\}$. The ciphertext is then

$$E_s^{\pm}(m, r, b_0, b_1) = ((-1)^{b_0} g^r \bmod n,$$
$$(-1)^{b_1} (h^{4\Delta^2 r} \bmod n)^{n^s} (n+1)^m \bmod n^{s+1})$$

Threshold Decryption: Given a ciphertext $c = (G, H) = E_s^{\pm}(m, r, b_0, b_1)$, it is only decrypted if the Jacobi symbol of G and H is 1. Since G is squared the d value can be computed as above. To decrypt H needs to be squared, so a slightly different computation is made:

$$H' = H^2 d^{-2n^s} = (n+1)^{2m} \bmod n^{s+1}$$

and the plaintext is found as $m = L_s(H')/2 \bmod n^s$.

Proving properties is now easier, since values can be squared to make sure they are in Q_n during the proof. In section 3.4 three proofs are shown: 1) a proof that something is a legel encryption, 2) a proof that something is a legal encryption of some publicly known plaintext, and 3) the threshold decryption proof. In appendix B some techniques for improving the complexity of most computations from $O(s^3k^3)$ to $O(s^2k^3)$ are shown.

3.4 Proof for the Proof Friendly Variant

Proof of Legal Encryption The purpose of the proof is to prove that given (G, H) there exist an $r \in \mathbb{Z}_N$ and an $m \in \mathbb{Z}_{n^s}$ such that $G = \pm g^r \bmod n$ and $H = \pm h^{4\Delta^2 r}(n+1)^m$.

Protocol for legal encryption
Input: $n, g, h, c = (G, H)$
Private input for P: $r \in \mathbb{Z}_N$ and $m \in \mathbb{Z}_{n^s}$, such that $c = E_s^{\pm}(m, r, b_0, b_1)$ for some b_0 and b_1.

1. P chooses at random r' in $\{0, ..., 2^{|N|+2k_2}\}$ and $m' \in \mathbb{Z}_{n^s}$, where k_2 is a secondary security parameter (e.g. 160 bits). P sends $c' = (G', H') = E_s^{\pm}(m', r', 0, 0)$ to V.
2. V chooses e, a random k_2 bit number, and sends e to P.

3. P sends $\hat{r} = r' + er$ and $\hat{m} = m' + em \bmod n^s$ to V. V checks that G, H, G', H' are prime to n, have Jacobi symbol 1 and that $E_s^{\pm}(2\hat{m}, 2\hat{r}, 0, 0) = (G'^2 G^{2e} \bmod n, H'^2 H^{2e} \bmod n^{s+1}) = c'^2 c^{2e}$, and accepts if and only if this is the case.

The protocol above can be proven to be sound and complete honest verifier zero-knowledge. This is enough for the election protocol in section 5, since it will only be used in an non-interactive setting using the Fiat-Shamir Heuristic and hash function $\mathcal{H}$ to generate the challenge $e = \mathcal{H}(G, H, G', H')$.

Proof of Legal Encryption of Certain Plaintext To protocol for legal encryptions, can be altered to a protocol for proving that something is a legal encryption of a certain plaintext m in the following way:

Protocol for legal encryption of message m
Input: $n, g, h, c = (G, H), m \in \mathbb{Z}_{n^s}$
Private input for P: $r \in \mathbb{Z}_N$, such that $c = E_s^{\pm}(m, r, b_0, b_1)$ for some b_0 and b_1.

1. P chooses at random r' in $\{0, ..., 2^{|N|+2k_2}\}$, where k_2 is a secondary security parameter (e.g. 160 bits). P sends $c' = (G', H') = E_s^{\pm}(0, r', 0, 0)$ to V.
2. V chooses e, a random k_2 bit number, and sends e to P.
3. P sends $\hat{r} = r' + er$ to V. V checks that G, H, G', H' are prime to n, have Jacobi symbol 1 and that $E_s^{\pm}(2em, 2\hat{r}, 0, 0) = (G'^2 G^{2e} \bmod n, H'^2 H^{2e} \bmod n^{s+1}) = c'^2 c^{2e}$, and accepts if and only if this is the case.

This protocol is also sound and complete honest verifier zero-knowledge, which follows directly from the protocol above and the observation, that if c' is not the encryption of the plaintext 0 there is only one e that can satisfy the last equation.

Decryption Proof To make the decryption share, the server calculated the value

$$d_i = G^{2\Delta\alpha_i} \bmod n$$

The server needs to prove that this was indeed what it submitted, but we have to allow a possible factor of -1, so we accept that $d_i = \pm G^{2\Delta\alpha_i}$, which is why the value d_i^2 is used in the Lagrange interpolation. What needs to be proven is that

$$\alpha_i = \log_g(h_i) = \log_{G^{4\Delta}}(d_i^2) \bmod p'q'$$

This can be done using a proof identical to that of Shoup RSA Threshold signatures [9].

Proof: Given a hash function $\mathcal{H}$ that outputs a k_2 bit hash, pick a random $r \in \{0, ..., 2^{|n|+2k_2} - 1\}$ and calculate

$$\hat{g} = g^r \bmod n, \hat{G} = G^{4\Delta r} \bmod n,$$
$$c = \mathcal{H}(g, G^{4\Delta}, h_i, d_i^2, \hat{g}, \hat{G}), z = \alpha_i c + r$$

The proof is the pair (c, z).

Verification: For a proof to be accepted the following equation has to hold

$$c = \mathcal{H}(g, G^{4\Delta}, h_i, d_i^2, h_i^{-c} g^z \bmod n, d_i^{-2c} G^{4\Delta z} \bmod n)$$

This proof of correctness is sound and statistical zero-knowledge under the random oracle model. This is proven in Shoups paper on Practical Threshold signatures [9] and is therefore omitted here.

3.5 Security of the Threshold Cryptosystems

Due to its homomorphic properties, our basic cryptosystem cannot be chosen ciphertext secure, so we cannot hope to prove that the threshold version is chosen ciphertext secure either. However, we can show a result saying essentially that as long as the adversary does not control the ciphertexts being decrypted, the threshold decryption releases no information other than the plaintext.

Definition 2. *A chosen plaintext threshold adversary $\mathcal{A}$ runs in probabilistic polynomial time and can statically and actively corrupt $t < w/2$ of the servers. In addition, for any efficiently samplable distribution $\mathcal{D}$, he may request that a message m be chosen according to $\mathcal{D}$ and then to see a random ciphertext containing m be decrypted using the threshold decryption protocol. A threshold public-key cryptosystem is secure against such an adversary if his view can be simulated in probabilistic polynomial time given only the public key.*

Note that since $\mathcal{D}$ is arbitrary, this includes the case where the adversary chooses m himself. This type of security will be sufficient in the mix-net below. Using a more elaborate decryption protocol, where a ciphertext is randomized before it is decrypted, an even stronger property can be shown, namely that security of the threshold system is equivalent to security of the non-threshold version under any attack. We omit this due to space limitations.

Lemma 1. *The threshold cryptosystems are semantically secure under Conjectures 1 and 2. They are also secure against any chosen plaintext threshold adversary as defined above.*

Proof. The semantic security follows from theorem 1 since an encryption of m in the basic cryptosystem can be transformed into an encryption of $4\Delta^2 m \bmod n^s$ in the threshold systems by raising the last component to the $4\Delta^2$'th power and in the last system by multiplying with $(-1)^{b_0}$ and $(-1)^{b_1}$.

The proofs of correctness used for the decryption shares are identical to the Shoup verification proofs for signatures shares in [9], where they were proved sound and statistical zero-knowledge. Furthermore, the secret sharing of the secret key is identical to the one used in [9]. Hence, it is straightforward to construct a simulation proof of security along the lines of the proof in [9]. A brief sketch: Given the public key, we can simulate the shares of corrupt players by choosing random values. We can then reconstruct the verification values of honest servers using Lagrange interpolation "in the exponent". To simulate the adversary's view of the decryption protocol, we choose m according to $\mathcal{D}$ and construct a random ciphertext containing m. In particular this means we know the relevant value of h^r. Given this and the shares of corrupt players, we can compute, with a statistically close distribution, the contribution from honest servers to the decryption. Their proofs of correctness can be simulated. By soundness of the proofs, the adversary will not be able to contribute bad values, so the decryption will output m, as desired. □

3.6 Homomorphic Properties

The 3 above cryptosystems are all additive homomorphic, which means that 2 encryptions can be combined to create a new encryption of the sum of the plaintexts in the original encryptions. The proof friendly cryptosystem can be shown to be additive homomorphic in the following way, which also works for the other 2 cryptosystems:

$$\begin{aligned}
E_s^{\pm}(m_0, r_0, b_{00}, b_{01}) & E_s^{\pm}(m_1, r_1, b_{10}, b_{11}) \\
&= (G_0, H_0)(G_1, H_1) \\
&= (G_0 G_1, H_0 H_1) \\
&= E_s^{\pm}(m_0 + m_1, r_0 + r_1, b_{00} \oplus b_{10}, b_{01} \oplus b_{11})
\end{aligned}$$

Note that this also provides an easy way to re-randomize an encryption:

$$E_s^{\pm}(m, r, b_0, b_1) E_s^{\pm}(0, r', b'_0, b'_1) = E_s^{\pm}(m, r + r', b_0 \oplus b'_0, b_1 \oplus b'_1)$$

The homomorphism only works when the 2 encryptions use the same s. To get around this, one of the following transformations can be used either to increase or to decrease the s. Both however will change the message inside the encryption.

Given an encryption $(G, H) = E_s^{\pm}(m, r, b_0, b_1)$, we can transform it into an encryption using $s' > s$ by doing the following transformation:

$$\begin{aligned}
(G', H') &= (G, H^{n^{s'-s}} \bmod n^{s'+1}) \\
&= ((-1)^{b_0} g^r, ((-1)^{b_1} (h^r)^{n^s} (n+1)^m)^{n^{s'-s}} \bmod n^{s'+1}) \\
&= ((-1)^{b_0} g^r, (-1)^{b_1} (h^r)^{n^{s'}} (n+1)^{m n^{s'-s}} \bmod n^{s'+1}) \\
&= E_{s'}^{\pm}(m n^{s'-s}, r, b_0, b_1)
\end{aligned}$$

If the encryption $(G, H) = E_s^{\pm}(m, r, b_0, b_1)$ needs to be transformed into an encryption using $s' < s$, we can do the following:

$$\begin{aligned}
(G', H') &= (G^{n^{s-s'}} \bmod n, H \bmod n^{s'+1}) \\
&= (((-1)^{b_0} g^r)^{n^{s-s'}} \bmod n, \\
&\qquad (-1)^{b_1} (h^r)^{n^s} (n+1)^{m \bmod n^{s'}} \bmod n^{s'+1}) \\
&= ((-1)^{b_0} g^{r n^{s-s'}} \bmod n, \\
&\qquad (-1)^{b_1} (h^{r n^{s-s'}})^{n^{s'}} (n+1)^{m \bmod n^{s'}} \bmod n^{s'+1}) \\
&= E_{s'}^{\pm}(m \bmod n^{s'}, r n^{s-s'}, b_0, b_1)
\end{aligned}$$

Since the order of g is relatively prime with n the value $rn^{s-s'}$ will span just as big a subgroup of $< g >$ as r alone.

4 Verifiable Length-Flexible Mix-net

4.1 The Mix-net Model

A mix-net is a network of servers that receive a number of encryptions, perform a random permutation of these, and output the plaintexts of the encryptions. This is done in such a way, that unless all servers (or most in some schemes) cooperate no one can link the input encryptions to the output plaintexts.

Note that for any single mix of messages, the inputs for the servers must be of the same length, since otherwise one could match the sizes of the inputs and outputs to find some information on the permutation. For practical applications this means, that a fixed upper bound will have to be decided for each mix, and all input messages for that mix have to be of the chosen size. This bound can chosen freely for each mix, however.

4.2 Adversaries

An adversary in [6] is defined by $(t_u, t_s)^{**}$, where the $*$ is either A for an active adversary or P for a passive adversary. The thresholds t_u and t_s are the maximal number of users and servers respectively, that can be controlled by the adversary. For example $(t_u, t_s)^{AP}$-adversary means that the adversary can read and change any value for up to t_u users and view any value inside t_s servers. A passive adversary only observes the values passing a server or user, but does not try to induce values into the process. An active adversary can attack the protocol by changing any value or refuse to supply results in any part of the protocol. The adversary is assumed to be non-adaptive, meaning that the users and servers being controlled by the adversary are decided in advance.

The mix-net in this paper is safe against these adversaries of increasing strength (u is the number of users and w the number of servers):

- $(u-2, w-1)^{PP}$-adversary: Here the adversary can see any value passing through all but 1 server and all but 2 of the users.

- $(u-2, w-1)^{AP}$-adversary: The adversary can see any value passing through all but 1 server and see and change any value inside all but 2 users.
- $(u-2, \lfloor (w-1)/2 \rfloor)^{AA}$-adversary: This is the strongest adversary that can see and change any value passing through all but 2 users and less than half of the servers.

Compared to the length-flexible mix-net in [6], the 2 first adversaries are the same, but the last one is improved from $(u-2, O(\sqrt{w}))^{AA}$ to $(u-2, \lfloor (w-1)/2 \rfloor)^{AA}$.

4.3 Security of the Mix-net

We will use a strong version of correctness, so even if users are working together with servers, they will not be able to change the message once the mix has started.

Definition 3 (Strong Correctness). *Given x encrypted messages as input, where y of the encryptions are malformed. The mix-net will output a permutation of the $x-y$ messages with correct decryptions, and discard all y malformed encryptions.*

Definition 4 (Anonymity). *Given a mix of x messages, and an $(t_u, t_s)^{**}$-adversary. Then the adversary should be unable to link any of the $x - t_u$ messages with any of the $x - t_u$ uncorrupted users who sent them.*

Definition 5 (Universal Verifiability). *Given the public view of the protocol being all the information written to the bulletin board, there exist a poly-time algorithm V that accepts only if the output of the protocol is correct, and otherwise outputs reject.*

Definition 6 (Robustness). *Given an $(t_u, t_s)^{*A}$-adversary the protocol should always output a correct result.*

The mix-network presented can be shown to satisfy these definitions under the different adversaries.

Theorem 3. *The basic mix-network provides strong correctness and anonymity (and robustness) against an $(u-2, w-1)^{*P}$-adversary, where u is the number of users and w the number of servers.*

Theorem 4. *The mix-network with threshold decryption provides strong correctness, anonymity, universal verifiability and robustness against an* $(u-2, \lfloor (w-1)/2 \rfloor)^{*A}$*-adversary, where* u *is the number of users and* w *the number of servers.*

4.4 The System

It is assumed that all communication in the protocol goes through a bulletin board, that anyone can access to verify the result. This means that the public key, and all outputs and proofs from the mix servers are written to this bulletin board.

The mix-network can be built from the threshold cryptosystem in the following way:

Key Generation: A trusted third party (TTP) generates $n = pq$ (as above) and $g \in Q_n$. Depending on the model the server picks the secrets the following way.
- **Passive adversary model:** For each mix server ($0 < i \leq w$) the TTP picks a random value $\alpha_i \in \mathbb{Z}_\tau$ and sets $\alpha = \sum_{0<i\leq w} \alpha_i \bmod \tau$. The public value is computed as $h = g^\alpha \bmod n$. The public key posted is (n, g, h) and the private key of server i is α_i.
- **Active adversary model:** Here, the key generation takes place exactly as described above for the threshold cryptosystem. The public key (n, g, h) and the verification values $(h_1, ..., h_w)$ are posted to the bulletin board. The private key α_i is given to the i'th server in a secure way.

Encryption: The s have to be fixed for each mix, so given a $m \in \mathbb{Z}_{n^s}$, random values $r \in \mathbb{Z}_N, b_0, b_1 \in \{0,1\}$ are chosen. The ciphertext posted on the bulletin board is

$$E_s^{\pm}(m, r, b_0, b_1) = ((-1)^{b_0} g^r \bmod n, \\ (-1)^{b_1} (h^{4\Delta^2 r} \bmod n)^{n^s} (n+1)^m \bmod n^{s+1})$$

Mixing phase: Before the mixing begins any ciphertext (G, H), where either G or H has Jacobi symbol -1 will be discarded as being incorrect. If an illegal ciphertext with Jacobi symbol 1 have been submitted it will be caught during decryption. Next set $I = \{1, ..., w\}$. While $I \neq \emptyset$ pick an $i \in I$ and let the i'th server make its mix permutation on the last correct output posted on the bulletin board:

- **Passive adversary model:** Since the adversary is passive, the mix server just do a random permutation and output a re-encryption for each of the ciphertexts (G, H) using the random values b_0, b_1, r:

$$\begin{aligned}(G', H') &= (G(-1)^{b_0} g^r \bmod n, \\ &\qquad H(-1)^{b_1}(h^{4\Delta^2 r} \bmod n)^{n^s} \bmod n^{s+1}) \\ &= (G, H) E_s^{\pm}(0, r, b_0, b_1)\end{aligned}$$

- **Active adversary model:** Here verification is needed to satisfy the universal verifiability, correctness and robustness of the system. To do this, the server picks a random permutation and creates a network of binary gates using the Waksman construction [1]. This network consists of $O(u \log(u))$ binary gates and can create any permutation of the inputs. For each binary gate a bit B is defined (and $\bar{B} = 1 - B$), determining if the gate should pass the encryptions straight through the gate or switch them, depending on the complete permutation of the mix. Each gate also has 2 ciphertexts (G_0, H_0) and (G_1, H_1) as input. The server chooses 6 random values: $x_0, x_1 \in \mathbb{Z}_N$ and $b_{00}, b_{01}, b_{10}, b_{11} \in \{0, 1\}$, and sets the 2 output ciphertexts for the gate to

$$\begin{aligned}(G'_B, H'_B) &= (G_0(-1)^{b_{00}} g^{x_0}, H_0(-1)^{b_{10}}(h^{4\Delta^2 x_0})^{n^s}) \\ &= (G_0, H_0) E_s^{\pm}(0, x_0, b_{00}, b_{10}) \\ (G'_{\bar{B}}, H'_{\bar{B}}) &= (G_1(-1)^{b_{01}} g^{x_1}, H_1(-1)^{b_{11}}(h^{4\Delta^2 x_1})^{n^s}) \\ &= (G_1, H_1) E_s^{\pm}(0, x_1, b_{01}, b_{11})\end{aligned}$$

  To prove this is done correctly the server needs to prove that the B, satisfying the 2 equations above, really exist. This can be done by showing that the difference between (G'_B, H'_B) and (G_0, H_0) is a legal encryption of 0 (and likewise for $(G'_{\bar{B}}, H'_{\bar{B}})$ and (G_1, H_1)) for some $B \in \{0, 1\}$. This can be done by using 4 concurrent runs of the legal encryption of the message 0 protocol using the technique from [3], that simulates the one of the 2 statements it is unable to answer truthfully:

$$(G'_0 G_0^{-1}, H'_0 H_0^{-1}) \text{ and } (G'_1 G_1^{-1}, H'_1 H_1^{-1})$$

are legal encryptions. of 0

or

$$(G'_1 G_0^{-1}, H'_1 H_0^{-1}) \text{ and } (G'_0 G_1^{-1}, H'_0 H_1^{-1})$$

are legal encryptions. of 0

These proofs are posted to the bullitin board along with the outputs and intermediate encryptions. If the proof of the mix is incorrect or the server refuses to post a complete mix, any output from the mix server is simply ignored, and the same input is used again for the next mix server.

When the mix is over or if the server refuses to output a mix, the server is removed from the set $I := I \backslash \{i\}$.

Decryption: After the mixing has been performed the decryption of each of the output ciphertexts (G, H) needs to be performed. The removal of the h^r part is different depending on the model and is achieved the following way

- **Passive adversary model:** The servers perform a decryption by each calculating their decryption part $d_i = G^{\alpha_i} \bmod n$. These values are removed from the encryption

$$H' = (H(\prod_{0<i\leq w} d_i)^{-n^s})^2 = \pm(n+1)^{2m} \bmod n^{s+1}$$

- **Active adversary model:** The servers check that at least $t+1$ servers have performed a legal mix, in which case at least 1 of them is honest, and it is safe to decrypt the encryptions. The value H' is computed as in the proof friendly threshold decryption in section 3.3.

In both the passive and active model the value H' has the form $\pm(n+1)^{2m}$ if the input was a correct encryption. If it was not a correct encryption it will have a power of h remaining: $H' = (h^r)^{n^s}(n+1)^{2m}$. This is the case if and only if $n \nmid H' - 1$ (since $n | (n+1)^{2m} - 1$) and the decryption is aborted in this case. Otherwise the message is decrypted as $m = L_s(H' \bmod n^{s+1})/2 \bmod n^s$.

The order of the mix servers can be chosen arbitrarily, which means that if server i is unavailable when it is supposed to mix, the server $i+1$ can do its mix. When server i gets back again it can perform its mix on the last output as if nothing had happened.

In appendix B a method is shown, that optimizes all computations except the last public exponentiation, from using $O(s^3k^3)$ time to only $O(s^2k^3)$.

4.5 Security Proofs

Proof sketch of theorem 3 and 4: The anonymity follows from the semantic security of the encryption scheme, which ensures that if just one server is honest, the adversary cannot track messages passing through this server. Furthermore, once this has happened, the ciphertexts being processed can no longer be controlled by the adversary. We can therefore apply Lemma 1 to show that the decryption phase releases no side information to the adversary. Warning: many technical details are omitted here due to space limitations.

Robustness is a result of the setup of the threshold decryption. If any server refuses to do the mix they're simply ignored and the result can always be decrypted using the threshold decryption.

Strong correctness comes from the zero-knowledge proofs, which ensure that the mix servers cannot change the message or tamper with the correctness of the encryption.

The use of the bulletin board model with verification proofs at all steps of the protocol ensures universal verifiability.

5 Self-Tallying Elections with Perfect Ballot Secrecy

In this section it will be shown how to make a more efficient self-tallying elections with perfect ballot secrecy based on the cryptosystem introduced in this paper. Note that for all practical purposes $s=1$ will be sufficient for this application. This will only be a brief walk-through of the technical details, so for a more in-depth explanation the reader is referred to [17].

The system uses the bulletin board model and it is assumed that a safe prime product n and a generator $g \in Q_n$ is setup in advance[1].

The modulus n can be generated once and for all. One option is to let a trusted third party do this. Note that since the factorization of n is never needed, not even in shared form, a trusted party solution can be quite acceptable, for instance one could use a secure hardware box that is destroyed after n has been generated.

Another option is to use a distributed protocol such as [16] or a generic multiparty computation. Note that protocols for this purpose can be set up such that no proper subset of the players can find the factors of n, in other words, we still ensure perfect ballot secrecy even if the players generate n themselves. This comes at the expense of possibly having to restart the key generation if faults occur, but this cost cannot be avoided if we need to handle dishonest majorities and is consistent with the way corrective fault tolerance is defined in [17].

Finally, at the expense of more work in the election itself, it is even posible to generate n simply as random number large enough so that it will be hard to factor completely. We postpone the details of this to the final version of the paper.

The element g can be generated by jointly generating some random value $x \in \mathbb{Z}_n^*$ and then defining g as $g = x^2$ which will be in Q_n.

The bulletin board also participates in the protocol to ensure that none of the actual voters will know the result before they vote. With a self-tallying scheme, this type of fairness cannot be achieved without such trust (see [17]). One may think of the bulletin board as a party that must vote 0 (so it will not influence the result), and is trusted to submit its vote only after all players have voted. The bulletin board however does not have to participate in every step of the protocol. It will only participate in: 1) the registration phase, where it registers its public key, 2) the error correction of the ballot casting, where it has some encrypted values it needs to reveal, and 3) the post ballot casting step, where it reveals its 0 vote, thereby enabling everyone to calculate the result.

---

[1] Accepting some extra conjectures, the techniques of [12] can be employed to in order to use any RSA modulus n where $\phi(n)/4$ is prime to Δ

5.1 Setup Phase

The setup phase consists of two tasks. First the voter registration and then the initialization of the voting system itself. In the registration phase voters, that want to participate in the election, register on the bulletin board. After all voters are registered, the voters need to setup the values to be used in the protocol. Since voters can be malicious in this part of the protocol there is an error correction step to correct any problems encountered.

Voter Registration Voter i chooses the private key α_i at random in $\mathbb{Z}_N$ and computes the value $h_i = g^{\alpha_i} \bmod n$. The voter registers by posting the public key $pk_i = (g, h_i)$ on the bulletin board. Let R be the set of all registered voters and for simplicity let's assume that $R = \{1, 2, ..., u\}$. To ensure fairness the bulletin board also generates a public key pk_0 and posts it on the bulletin board (we set $R_0 = R \cup \{0\}$).

Initialization Each voter $i \in R$ picks random values $s_{ij} \in \mathbb{Z}_{n^s}$ for each $j \in R$ and random $r_{ij} \in \mathbb{Z}_N$ for each $j \in R_0$. The value s_{i0} is set to $-\sum_{j \in R} s_{ij} \bmod n^s$, which ensures that $\sum_{j \in R_0} s_{ij} = 0 \bmod n^s$.

The voter i publishes the encryptions

$$c_{ij} = (G_{ij}, H_{ij}) = E^{\pm}_{s,pk_j}(s_{ij}, r_{ij})$$

for all $j \in R_0$ along with a proof, that these are indeed legal encryptions and the sum of the plaintexts is 0 modulo n^s.

To prove these are legal encryptions, the proof from section 3.4 is used. To prove that the sum of the plaintexts in the encryptions $(G_{i0}, H_{i0}), ..., (G_{iu}, H_{iu})$ is 0, it is enough to look at the product $H_{i0} \cdots H_{0u}$. The resulting value is

$$H_{i0} \cdots H_{iu} = (h_0^{r_{i0}} \cdots h_u^{r_{iu}})^{n^s} (n+1)^{s_{i0}+\cdots+s_{iu}}$$

which is an n^s'th power iff $\sum_{j \in R_0} s_{ij} = 0 \bmod n^s$. The protocol for n^s'th powers from [11] can be used to prove this, since the voter knows an n^s'th root of this number, namely $h_0^{r_{i0}} \cdots h_u^{r_{iu}}$.

Error Correction of the Initialization Let Q_1 be the set of voters that either doesn't supply all the encryptions or supply invalid proofs. Any values submitted by voters in Q_1 are simply ignored. The values that the honest voters created for the voters in Q_1 will remain unused, which is a problem since the numbers should sum to 0. To correct this, the honest voters open all encryptions assigned to voters in Q_1.

More formally for all $i \in R \backslash Q_1$ the voter i releases the values s_{ij}, r_{ij} for all $j \in Q_1$. Since these values are uniquely determined by the encryption, this step can be verified by checking that $c_{ij} = E^{\pm}_{s,pk_j}(s_{ij}, r_{ij})$. Should a voter refuse to publish this information they are simply added to Q_1 and their values are revealed.

5.2 Ballot Casting

Ballot Casting Each voter $j \in R \backslash Q_1$ retrieves the encryptions c_{ij} $\forall i \in R \backslash Q_1$ and combines them:

$$c_j = \prod_{\forall i \in R \backslash Q_1} c_{ij} = E^{\pm}_{s,pk_j}(\sum_{\forall i \in R \backslash Q_1} s_{ij}, r)$$

for some value of r. Voter j decrypts c_j using the private key α_j to get $t_j = \sum_{\forall i \in R \backslash Q_1} s_{ij}$.

Voter j then submits the values $d_j = E^{\pm}_{s,pk_j}(v_j, r_j)$ and $x_j = v_j + t_j$, where v_j is the value representing the candidate voter j votes for, say 0 or 1 for a yes/no election; the value $r_j \in \mathbb{Z}_N$ is chosen at random. The easiest way to understand this is to note that if we ignore the error correction (i.e., assume that no faults occur), then the t_j's will be a set of random numbers that sum to 0. So if we can ensure that x_j was formed by adding an allowable value of v_j to t_j, then $res = \sum_j x_j$ will be the election result, i.e., the sum of the v_j 's. Moreover, the randomness of the t_j ensures that given the x_j's, all possible sets of v_j's summing to res are equally likely.

To prove that d_j is a legal encryption of an allowable value of v_j, the proof from section 3.3 can be used to ensure that it is a correct encryption, and the proof of a legal vote value in [11] (which is logarithmic in the number of candidates) can be used on the second value in the encryption to prove that a legal v_j have been encrypted.

To prove that d_j is an encryption of the same v_j, that was used to make x_j, the voter proves that

$$d_j c_j E^{\pm}_{s,pk_j}(x_j, 0)^{-1} = E^{\pm}_{s,pk_j}(0, r')$$

for some given r' using the n^s'th power proof from [11], and sends this proof to the bulletin board. This time the required n^s'th root can be computed as: $h_j^{r_j} \cdot (\prod G_{ij}^{4\Delta^2 \alpha_j}) \cdot 1$. This holds iff the same v_j is used in d_j and x_j since

$$\begin{aligned} d_j c_j (E^{\pm}_{s,pk_j}(v_j + t_j, 0))^{-1} &= E^{\pm}_{s,pk_j}(v_j, r_j) E^{\pm}_{s,pk_j}(t_j, r) E^{\pm}_{s,pk_j}(-(x_j), 0) \\ &= E^{\pm}_{s,pk_j}(v_j + t_j - (v_j + t_j), r') \\ &= E^{\pm}_{s,pk_j}(0, r') \end{aligned}$$

Error Correction of Ballot Casting Let Q_2 be the set of voters disqualified during the ballot casting. Again there are some values that will not be used by the voters in Q_2, and these are simply published on the bulletin board as in the error correction of the initialization.

However this time the values created by voters in Q_2 have been used by the honest voters (for any $i \in Q_2$ the value of s_{ii} is only know by i, and all honest voters j have used s_{ij}). To correct this the values have to be published, but the secret values r_{ij} are unknown to j. So for all $i \in Q_2$ each $j \in R_0 \backslash (Q_1 \cup Q_2)$ (voters and bulletin board) decrypts and reveals the plaintext of c_{ij}, which is s_{ij} and proves that

$$c_{ij} E^{\pm}_{s,pk_j}(s_{ij}, 0)^{-1} = E^{\pm}_{s,pk_j}(0, r)$$

using the n^s'th power proof from [11]. This time the required n^s'th root is: $G_{ij}^{4\Delta^2 \alpha_j} \cdot 1$. Should anyone refuse to participate in the error correction they're simply added to Q_2 and their values published as before.

Now let $Q_{bad} = Q_1 \cup Q_2$ denote all voters that have been removed in the error correction steps, and let $R_{good} = R \backslash Q_{bad}$ be the voters that completed the whole protocol honestly.

Post Ballot Casting When the ballot phase is over, and all parties have either submitted their vote or been removed using the error correction, the bulletin board computes

$$c_0 = \prod_{\forall i \in R \setminus Q_{bad}} c_{i0} = E^{\pm}_{s,pk_0}(\sum_{\forall i \in R \setminus Q_{bad}} s_{i0}, r)$$

for some r and gets the plaintext $t_0 = \sum_{\forall i \in R \setminus Q_{bad}} s_{i0}$ by decrypting c_0. The bulletin board then posts t_0 along with a proof that $c_0 E^{\pm}_{s,pk_0}(t_0, 0)^{-1}$ is a n^s'th power according to the proof in [11] (the value is calculated as $(\prod G_{i0}^{4\Delta^2 \alpha_0}) \cdot 1$).

5.3 Tallying

At this point the result can be computed as:

$$\begin{aligned} res &= t_0 + \sum_{j \in R_{good}} x_j + \sum_{i \in R_{good}, j \in Q_{bad}} s_{ij} - \sum_{i \in Q_2, j \in R_{good} \cup \{0\}} s_{ij} \\ &= \sum_{j \in R_{good}} v_i \bmod n^s \end{aligned}$$

The first sum is all the x_j values that have been posted on the bulletin board according to protocol. The values in the second sum are the values of the disqualified voters that where revealed in the error correction of the initialization and the first value revealed in the error correction of the ballot casting. The third sum is the sum of the second values revealed in the error correction of the ballot casting.

For lack of space, we do not give formal definitions and proofs of security here. However, perfect ballot secrecy follows since first, we can ignore those encryptions that remain unopened, due to the semantic security. What remains are the public numbers: x_i's, and the numbers revealed during error correction. One then observes that for any corrupted subset and given result, all possible sets of votes from honest players leading to the given result are equally likely. The correctness follows from the following observations:

$$R \setminus Q_1 = R_{good} \cup Q_2 \quad \text{and} \quad R = Q_{bad} \cup R_{good}$$

which implies (using that only c_{ij} for $i \in R \backslash Q_1$ was used to get t_j)

$$\begin{aligned} & t_0 + \sum_{j \in R_{good}} x_j \\ & \quad = \sum_{j \in R_{good}} v_j + \sum_{j \in R_{good} \cup \{0\}} t_j \\ & \quad = \sum_{j \in R_{good}} v_j + \sum_{i \in R_{good}, j \in R_{good} \cup \{0\}} s_{ij} + \sum_{i \in Q_2, j \in R_{good} \cup \{0\}} s_{ij} \bmod n^s \end{aligned}$$

and now using $\forall i \in R_{good}$: $\sum_{j \in R_0} s_{ij} = 0 \bmod n^s$ we get

$$\begin{aligned} & t_0 + \sum_{j \in R_{good}} x_j + \sum_{i \in R_{good}, j \in Q_{bad}} s_{ij} \\ & \quad = \sum_{j \in R_{good}} v_j + \sum_{i \in R_{good}, j \in R_0} s_{ij} + \sum_{i \in Q_2, j \in R_{good} \cup \{0\}} s_{ij} \\ & \quad = \sum_{j \in R_{good}} v_j + \sum_{i \in Q_2, j \in R_{good} \cup \{0\}} s_{ij} \bmod n^s \end{aligned}$$

now subtracting the last sum we get the wanted result

$$\begin{aligned} res &= t_0 + \sum_{j \in R_{good}} x_j + \sum_{i \in R_{good}, j \in Q_{bad}} s_{ij} - \sum_{i \in Q_2, j \in R_{good} \cup \{0\}} s_{ij} \\ &= \sum_{j \in R_{good}} v_j \bmod n^s \end{aligned}$$

5.4 Efficiency Comparison to Scheme from [17]

The work of the 2 schemes are comparable in all steps of the protocol except in the tallying phase. Here the protocol of [17] needs to do an exhaustive search in a space of size $2u$, which can be optimized to $O(\sqrt{u})$ multiplications. However, the protocol above obtains the result of the election by simply adding the values posted to the bulletin board.

Our scheme generalizes to multi-candidate elections in exactly the same way as [11]. In particular, the tallying phase remains at the same number of additions. For the scheme from [17], the search for the result would take $\Omega((\sqrt{u})^l)$ multiplications for l candidates.

References

1. A. Waksman: *A permutation network*, Journal of the ACM 15(1), January 1968, pp. 159-163.
2. A. Shamir: *How to Share a Secret*, Communications of the ACM 22(11), November 1979, pp. 612-613.
3. R. Cramer, I. Damgård and B. Schoenmakers: *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, Proceedings of Crypto '94, Springer-Verlag LNCS 839, pp. 174-187.
4. M. Abe: *Mix-networks on Permutation Networks*, Proceedings of AsiaCrypt '99, Springer Verlag LNCS 1716, pp. 258-273.
5. P. Paillier: *Public-Key Cryptosystems based on Composite Degree Residue Classes*, Proceedings of EuroCrypt '99, Springer Verlag LNCS 1592, pp. 223-238.
6. M. Abe and M. Ohkubo: *A Length-Invariant Hybrid Mix*, Proceedings of AsiaCrypt 2000, Springer Verlag LNCS 1976, pp. 178-191.
7. Y. Desmedt and K. Kurosawa: *How to break a practical MIX and design a new one*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS 1807, pp. 557-572.
8. 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.
9. V. Shoup: *Practical Threshold Signatures*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS 1807, pp. 207-220.
10. M. Abe and F. Hoshino: *Remarks on Mix-network Based on Permutation Networks*, Proceedings of Public Key Cryptography 2001, Springer Verlag LNCS 1992, pp. 317-324.
11. 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 1992, pp. 119-136. Final version appears as [14].
12. I. Damgård and M. Koprowski: *Practical Threshold RSA Signatures Without a Trusted Dealer*, Proceedings of EuroCrypt 2001, Springer Verlag LNCS 2045, pp. 152-165.
13. M. Jakobsson and A. Juels, *An optimally robust hybrid mix network*, Annual ACM Symposium on Principles of Distributed Computing 2001, pp 284-292.
14. I. Damgård, M. Jurik and J.B. Nielsen: *A Generalization of Paillier's Public-Key System with Applications to Electronic Voting*, to appear in International Journal of Information Security, Springer Verlag.
15. R. Cramer and V. Shoup: *Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption*, Proceedings of EuroCrypt 2002, Springer Verlag LNCS 2332, pp. 45-64.
16. J. Algesheimer, J. Camenisch and V. Shoup: *Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products* Proceedings of Crypto 2002, Springer Verlag LNCS 2442, pp. 417-432.
17. A. Kiayias and M. Yung: *Self-Tallying Elections and Perfect Ballot Secrecy*, Proceedings of Public Key Cryptography 2002, Springer Verlag LNCS 2274, pp. 141-158.

Adversary $\mathcal{A}'$:

1. Get public key n and forward it to $\mathcal{A}$.
2. Get message m from $\mathcal{A}$.
3. Return on m.
4. Get the encryption $c = E_s(\hat{m}, r)$, where
   - $\hat{m} = m$ or
   - $\hat{m}$ uniformly random in $\mathbb{Z}_{n^s}$.
5. Calculate $c' = (c(n+1)^{-m})^2(n+1)^m \bmod n^{s+1}$
6. Send c' to $\mathcal{A}$.
7. Get the bit b from $\mathcal{A}$
8. Return b.

Fig. 1. Algorithm for adversary $\mathcal{A}'$ breaking the Damgård-Jurik cryptosystem using the adversary $\mathcal{A}$.

A Security of the Cryptosystem

Before proving the security of the cryptosystem a lemma stating that using only Q_n does not degrade the security of the Damgård-Jurik cryptosystem.

Lemma 2. *The Damgård-Jurik cryptosystem using $r \in \mathrm{Q}_n$ is as semantically secure, with respect to definition 1, as the cryptosystem where $r \in \mathbb{Z}_n^*$.*

Proof. To show that the security is equivalent, assume an adversary $\mathcal{A}$ exits, that can break the semantically security of the quadratic cryptosystem. Then the adversary $\mathcal{A}'$ shown in figure 1 breaks the original cryptosystem.

Given an encryption of the message m, the ciphertext c' generated by $\mathcal{A}'$ will be a legal encryption of m with an uniform distribution of the random r, which follows from:

$$\begin{aligned}(c(n+1)^{-m})^2(n+1)^m &= (r^{n^s}(n+1)^{m-m})^2(n+1)^m \\ &= (r^2)^{n^s}(n+1)^m \\ &= E_s(m, r^2) \bmod n^{s+1}\end{aligned}$$

In the case that $\hat{m}$ is chosen uniformly random from the message space $\mathbb{Z}_{n^s}$ the resulting c' is

$$\begin{aligned}(c(n+1)^{-m})^2(n+1)^m &= (r^{n^s}(n+1)^{\hat{m}-m})^2(n+1)^m \\ &= r^{2^{n^s}}(n+1)^{2\hat{m}-m} \\ &= E_s(2\hat{m}-m, r^2) \bmod n^{s+1}\end{aligned}$$

Because $\gcd(2, n) = 1$, the function $2\hat{m} - m$ is a 1-1 permutation of $\mathbb{Z}_{n^s}$. So the new encryption will be uniformly random in $\mathbb{Z}_{n^s}$ since $\hat{m}$ is. The probabilities for $\mathcal{A}'$ are $p_0(\mathcal{A}', k) = p_0(\mathcal{A}, k)$ and $p_1(\mathcal{A}', k) = p_1(\mathcal{A}, k)$, which means that the advantage of $\mathcal{A}'$ is the same as for $\mathcal{A}$. $\square$

Given the lemma it is easy to prove that the dual key cryptosystem is semantically secure.

Theorem 5 (Semantically Security). *Under the conjectures 1 (DCRA) and 2 (composite-DDH) the cryptosystem is semantically secure with respect to definition 1.*

Proof. The proof is done in a 3 step hybrid reduction using the composite-DDH conjecture and the semantical security of the Quadratic variant of the Damgård-Jurik cryptosystem (DCRA). Given the public key (n, g, h), the following 4 pairs are indistinguishable under conjecture 1 and 2:

1. $(g^k \bmod n, (h^k)^{n^s}(n+1)^m \bmod n^{s+1})$
2. $(g^k \bmod n, (r)^{n^s}(n+1)^m \bmod n^{s+1})$, where r is uniformly random in Q_n
3. $(g^k \bmod n, (r)^{n^s}(n+1)^{m'} \bmod n^{s+1})$, where m' is random in $\mathbb{Z}_{n^s}$
4. $(g^k \bmod n, (h^k)^{n^s}(n+1)^{m'} \bmod n^{s+1})$

If tuple 1 and 4 are indistinguishable then the dual key system is semantically secure following the definition. If an adversary can distinguish between tuple 1 and 4 with advantage $\epsilon > \frac{1}{f(k)}$ for some polynomial $f(k)$, there will be an adversary able to distinguish between a pair of consecutive tuples with probability larger than $\epsilon' > \frac{1}{3f(k)}$. Each of the 3 pairs are shown below to be indistinguishable, thereby showing that an adversary between the tuples 1 and 4 cannot exist.

The pairs 1 and 2 are indistinguishable due to a reduction to composite-DDH. Assuming an adversary $\mathcal{B}$ having a non-negligible advantage of distinguishing pairs 1 and 2, an adversary $\mathcal{B}'$ can be built that will break composite-DDH with the same advantage. The algorithm for the adversary $\mathcal{B}'$ can be seen in figure 2.

The 2 cases of y in the composite-DDH correspond to tuple 1 and 2 respectively:

Adversary $\mathcal{B}'$:

1. Get the composite-DDH tuple: (n, g, g^a, g^b, y).
2. Give the public key $(n, g, h := g^a)$ to $\mathcal{B}$.
3. Get message m from $\mathcal{B}$.
4. Give the encryption $(g^b, y^{n^s}(n+1)^m \bmod n^{s+1})$ to $\mathcal{B}$.
5. Get the bit b from $\mathcal{B}$.
6. Return b.

Fig. 2. Algorithm for adversary $\mathcal{B}'$ that break composite-DDH given adversary $\mathcal{B}$.

$\mathbf{y = g^{ab}}$**:** Here $y = g^{ab} = (g^a)^b = h^b$, which is the value used in tuple 1.

y uniformly random in $\mathrm{Q}_{\mathbf{n}}$**:** This is a random value as in tuple 2.

Thereby tuple 1 and 2 are indistinguishable under the composite-DDH assumption.

That pair 2 and 3 are indistinguishable follow directly from the fact that the quadratic Damgård-Jurik system is semantically secure under DCRA. The encryption $r^{n^s}(n+1)^m \bmod n^{s+1}$ of a message m is indistinguishable from the encryption of a random message $m' \in \mathbb{Z}_{n^s}$: $r^{n^s}(n+1)^{m'} \bmod n^{s+1}$.

The pairs 3 and 4 are indistinguishable following the same reduction as from 1 to 2. □

B Optimization of the cryptosystem

In the description of the cryptosystems (and the mix-net) the notation $(h^{4\Delta^2 r})^{n^s} \bmod n^{s+1}$ have been used for the simplicity of the scheme. However, the number of bit operations needed to calculate this is $O((sk)^3) = O(|msg|^3)$, where k is the security parameter of n and $|msg|$ is the length of the messages encrypted. It is however easy to transform this into something that can be computed more easily by using the following equality:

$$(h^{4\Delta^2 r})^{n^s} = (h^{4\Delta^2 n^s})^r \bmod n^{s+1}$$

The important thing to note is that the value $h^{4\Delta^2 n^s}$ is fixed for a given s. This means that the value $h_s^* = h^{4\Delta^2 n^s} \bmod n^{s+1}$ can be calculated by the decryption servers in advance and passed along with the public key.

Another nice attribute is that given the value h_s^*, it is possible in time $O(s^2k^3)$, to compute h_{s+1}^* by the following method:

$$h_{s+1}^* = (h_s^* \bmod n^{s+1})^n \bmod n^{s+2}$$

which works because

$$\begin{aligned}
(h_s^* \bmod n^{s+1})^n &= (h^{4\Delta^2 n^s} + kn^{s+1})^n \\
&= h^{4\Delta^2 n^{s+1}} + (h^{4\Delta^2 n^s})^{n-1}\binom{n}{1}kn^{s+1} \\
&\qquad + (h^{4\Delta^2 n^s})^{n-2}\binom{n}{2}k^2n^{2(s+1)} + ... \\
&= h^{4\Delta^2 n^{s+1}} \\
&= h_{s+1}^* \bmod n^{s+2}
\end{aligned}$$

This means that all exponentiations of h in the encryption and decryption (and for mixing, proving and verifying in the mix-net) can be reduced in complexity from $O(s^3k^3)$ to $O(s^2k^3)$ if the s' used is within a constant larger than a s used for a public generator value h_s^*.

The encryption also has an exponentiation of the form $(n+1)^m$, which can be computed using the formula

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

thus achieving complexity $O(s^3k^2)$ as shown in [14].

The only exponentiation that cannot be optimized is the last public exponentiation of the combined server decryption value. The reason is that the base of the exponentiation in this instance is not fixed.

Recent BRICS Report Series Publications

RS-03-16 **Ivan B. Damgård and Mads J. Jurik.** ***A Length-Flexible Threshold Cryptosystem with Applications*****. March 2003. 31 pp.**

RS-03-15 **Anna Ingólfsdóttir.** ***A Semantic Theory for Value–Passing Processes Based on the Late Approach*****. March 2003. 48 pp.**

RS-03-14 **Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard.** ***From Interpreter to Compiler and Virtual Machine: A Functional Derivation*****. March 2003. 36 pp.**

RS-03-13 **Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard.** ***A Functional Correspondence between Evaluators and Abstract Machines*****. March 2003. 28 pp.**

RS-03-12 **Mircea-Dan Hernest and Ulrich Kohlenbach.** ***A Complexity Analysis of Functional Interpretations*****. February 2003. 70 pp.**

RS-03-11 **Mads Sig Ager, Olivier Danvy, and Henning Korsholm Rohde.** ***Fast Partial Evaluation of Pattern Matching in Strings*****. February 2003. 14 pp. This report is superseded by the later report BRICS RS-03-20.**

RS-03-10 **Federico Crazzolara and Giuseppe Milicia.** ***Wireless Authentication in χ-Spaces*****. February 2003. 20 pp.**

RS-03-9 **Ivan B. Damgård and Gudmund Skovbjerg Frandsen.** ***An Extended Quadratic Frobenius Primality Test with Average and Worst Case Error Estimates*****. February 2003. 53 pp.**

RS-03-8 **Ivan B. Damgård and Gudmund Skovbjerg Frandsen.** ***Efficient Algorithms for gcd and Cubic Residuosity in the Ring of Eisenstein Integers*****. February 2003. 11 pp.**

RS-03-7 **Claus Brabrand, Michael I. Schwartzbach, and Mads Vanggaard.** ***The METAFRONT System: Extensible Parsing and Transformation*****. February 2003. 24 pp.**

RS-03-6 **Giuseppe Milicia and Vladimiro Sassone.** ***Jeeg: Temporal Constraints for the Synchronization of Concurrent Objects*****. February 2003. 41 pp. Short version appears in Fox and Getov, editors,** ***Joint ACM-ISCOPE Conference on Java Grande*****, JGI '02 Proceedings, 2002, pages 212–221.**