Cryptographic Counters and Applications to Electronic Voting

JONATHAN KATZ[1] STEVEN MYERS[2] RAFAIL OSTROVSKY[3]

[1] Telcordia Technologies and
Department of Computer Science, Columbia University.
jkatz@cs.columbia.edu

[2] Department of Computer Science, University of Toronto.
Work done while the author was at Telcordia Technologies.
myers@cs.toronto.edu

[3] Telcordia Technologies, Inc., 445 South Street, Morristown, NJ 07960.
rafail@research.telcordia.com

Abstract. We formalize the notion of a *cryptographic counter*, which allows a group of participants to increment and decrement a cryptographic representation of a (hidden) numerical value privately and robustly. The value of the counter can only be determined by a trusted authority (or group of authorities, which may include participants themselves), and participants cannot determine any information about the increment/decrement operations performed by other parties.

Previous *efficient* implementations of such counters have relied on fully-homomorphic encryption schemes; this is a relatively strong requirement which not all encryption schemes satisfy. We provide an alternate approach, starting with any encryption scheme homomorphic over the additive group $\mathbb{Z}_2$ (i.e., 1-bit XOR). As our main result, we show a general and efficient reduction from any such encryption scheme to a general cryptographic counter. Our main reduction does not use additional assumptions, is efficient, and gives a novel implementation of a general counter. The result can also be viewed as an efficient construction of a general n-bit cryptographic counter from any 1-bit counter which has the additional property that counters can be added securely.

As an example of the applicability of our construction, we present a cryptographic counter based on the quadratic residuosity assumption and use it to construct an efficient voting scheme which satisfies universal verifiability, privacy, and robustness.

1 Introduction

1.1 Cryptographic Counters

In this paper we present an efficient and secure protocol for calculating the sum of integers, where each integer is held privately by a single participant. Although it is clear that this can be achieved via the completeness results for multi-party computation (see [14] for a complete review of multi-party computation and related results), such constructions are only of theoretical interest as they are too inefficient to be of practical use. In order to construct our secure addition protocol, we introduce an abstraction we call a *cryptographic counter* that may be of independent interest. In particular, such counters may have a variety of applications, especially as subroutines in larger multi-party computations. We give a formal definition of cryptographic counters, and provide a construction based on any encryption scheme homomorphic over the additive group $\mathbb{Z}_2$.

Informally, a cryptographic counter is a public string which can be viewed as an encryption of a value such that the value is hidden from all participants except a trusted authority (who holds some secret key). Only the trusted authority can decrypt and thereby determine the value of the counter, whereas all participants have the ability to increment or decrement (*update*) the counter by an arbitrary amount. Information about updates (e.g., whether the counter was incremented or decremented) is kept hidden from all other participants. We also consider *restricted* cryptographic counters for which the set of legal update operations is constrained in some publicly-known way.

Previous constructions of cryptographic counters (in the context of voting schemes) have relied on what we call *fully-homomorphic* encryption. Informally, this is an encryption scheme for which, for any $n_0 > 0$, there is some choice of the security parameter such that the resulting encryption is homomorphic over (the additive group) $\mathbb{Z}_n$, where $n \geq n_0$. It is clear how a cryptographic counter can be constructed given this strong property (the difficult aspects of previous constructions were providing efficient proofs of validity and achieving threshold decryption). In this paper, we provide a construction of an n-bit cryptographic counter based on any 1-bit cryptographic counter that also allows secure addition (mod 2) of multiple counters. This immediately implies a construction from any encryption scheme homomorphic over $\mathbb{Z}_2$. As a concrete example, we present an *efficient* n-bit counter based only on the quadratic residuosity assumption.

Addition is a useful function to compute privately, as many of the currently-proposed applications of secure multi-party computation rely heavily on summing secret values held by different individuals. It has particular relevance to the problem of secure electronic voting, in which each participant holds a vote which is either 0 or 1, and the participants wish to determine the tally without revealing individual votes. As an example of the applicability of cryptographic counters, we use them to build a secure voting scheme and compare it to previously-proposed solutions. In particular, ours is the first efficient construction of a voting scheme which is not based on fully-homomorphic encryption.

1.2 Secure Electronic Voting

An electronic voting scheme is a protocol allowing voters to cast a vote by interacting with a set of authorities who collect the votes, tally them, and publish the final result. There are a variety of properties which may be desired of an electronic voting scheme; however, the cryptographic literature has traditionally focused on the following three requirements:

Privacy ensures that an individual's vote is kept hidden from (any reasonably-sized coalition of) other voters and even the authorities themselves.

Universal Verifiability means that any party, including a passive observer, can be convinced that all votes cast were valid and that the final tally was computed correctly.

Robustness guarantees that the final tally can be correctly computed even in the presence of faulty behavior of a number of parties.

It is furthermore desirable to minimize the interaction between parties. In particular, voters should not have to interact with each other to cast a vote or (ideally) to prove validity of votes, and the authorities should be able to remain off-line until the election is concluded. Other features are not considered in the present work. For example, information-theoretic privacy is sometimes required [8], while we only require computational privacy. Receipt-freeness [2] and preventing vote-duplication can be achieved by other means (see, for example, [17]) and are not considered here.

Many voting schemes meeting the above requirements have been proposed [6, 3, 4, 8, 9, 23, 10]. However, all previously-known schemes achieving universal verifiability rely on *fully-homomorphic* encryption schemes, where the homomorphism is over additive group $\mathbb{Z}_n$ and n is larger than the number of voters (our use of the term "fully-homomorphic" is explained above). One typical paradigm is as follows: say voter i wishes to cast vote v_i, where, for a valid vote, we have $v_i \in \{0, 1\}$. To vote, voter i publicly posts[1] $\mathcal{E}_{\text{pk}}(v_i)$, the encryption of v_i under some public key established by the set of authorities. When everyone has voted, the authorities compute the product of the encryptions (which can be publicly computed) and decrypt the result; this gives the correct final tally since:

$$\mathcal{D}_{\text{sk}}\left(\mathcal{E}_{\text{pk}}(v_1)\cdots\mathcal{E}_{\text{pk}}(v_N)\right) = v_1 + \cdots + v_N,$$

where equality holds by the homomorphic properties of the encryption scheme. Depending on the level of trust in the authorities, they may also provide a (publicly verifiable) proof that decryption was done correctly. In this way, everyone is assured that all votes were correctly counted.

Many examples of fully-homomorphic encryption schemes are known (for example: [12, 6, 21]). The voting schemes of [6, 3, 4] are based on the r-th residuosity assumption, those of [8, 9, 23] are based on the discrete logarithm assumption in prime groups, and the scheme of [10] is based on hardness of deciding residue classes in $\mathbb{Z}^*_{N^2}$. Even so, it is interesting to determine the minimal assumptions under which an efficient voting protocol can be constructed.

We show how privacy and universal verifiability can be achieved without fully-homomorphic encryption. Our construction uses an n-bit counter which, in turn, is constructed from any encryption scheme homomorphic over $\mathbb{Z}_2$ (i.e., the

[1] This might be accompanied by a proof of validity, but for simplicity we focus here on that portion of the protocol which relies on the homomorphic properties of the encryption.

	Size of Vote + Proof	Voter Computation	Authority Computation
[8]	$\mathcal{O}(k_1 M)$	$\mathcal{O}(k_1^3 M)$	$\mathcal{O}(k_1^3 L)$
[9]	$\mathcal{O}(k_1)$	$\mathcal{O}(k_1^3)$	$\mathcal{O}(k_1^3 L)$
Present work	$\mathcal{O}(k_1 k_2 \log L)$	$\mathcal{O}(k_1^2 k_2 \log L)$	$\mathcal{O}(k_1^2 \log L + L)$

Table 1. Efficiency of some voting schemes. L is the number of voters, M is the number of authorities, k_1 is a security parameter, and 2^{-k_2} is a bound on the probability of cheating (in [8, 9], the probability of cheating is 2^{-k_1}). Computation is measured in bitwise operations, assuming multiplication of k-bit numbers requires $\mathcal{O}(k^2)$ operations.

1-bit XOR operation). Using as a specific example the well-studied encryption scheme based on the hardness of deciding quadratic residuosity [16], we show how to achieve robustness as well.

Often, basing a result on a weaker assumption results in an impractical scheme. However, our resulting voting scheme is efficient enough to be practical. A comparison of the efficiency of our construction with those of [8, 9] appears in Table 1. Our simplest solution, while being both size- and computation-efficient, requires sequential execution and hence $\mathcal{O}(L)$ rounds (as compared with previous solutions which require $\mathcal{O}(1)$ rounds). We discuss ways of dealing with this issue in Section 5.

2 Definitions

In this section we formalize the notion of a cryptographic counter. Although related notions have been folklore in the cryptographic community (particularly in the context of electronic voting), a formal definition has, to the best of our knowledge, not previously appeared.

COUNTERS. In order to more easily define a *cryptographic* counter, we first need a formal definition of a counter.

Definition 1. *An n-*counter *consists of a set S along with a pair of algorithms (D, T) in which:*

- *$S = \{s_1, \ldots\}$ represents the* set of states *of the counter.*
- *D, the* decoding algorithm, *is a deterministic algorithm which takes as input a state $s \in S$ and returns a number $i \in \mathbb{Z}_n$. This defines a mapping from states in S to numbers in the range $[0, n-1]$.*
- *T, the* transition algorithm, *is a probabilistic algorithm which takes as input a state $s \in S$ and an integer $i \in \mathbb{Z}_n$ and returns a state $s' \in S$. This function defines legal update operations on the counter.*

We require that for all $s \in S$ and $i \in \mathbb{Z}_n$, if $s' \leftarrow T(s, i)$, then $D(s') = D(s) + i \bmod n$.

Note that subtraction of integer i can be done by simply computing the inverse of i in $\mathbb{Z}_n$ and adding $-i$ using the transition algorithm.

CRYPTOGRAPHIC COUNTERS. We now turn to the definition of a cryptographic counter. We first define its components, and follow this with definitions of security against two types of adversaries: honest-but-curious and malicious. All algorithms are assumed to run in time polynomial in the security parameter k, and n is fixed independently of k.

Definition 2. *A* cryptographic n-counter *is a triple of algorithms* $(\mathcal{G}, D, T)$ *in which:*

- $\mathcal{G}$*, the* key generation algorithm*, is a probabilistic algorithm that on input* 1^k *outputs a public key/secret key pair* (pk,sk) *and a string* s_0*. The secret key, in turn, implicitly defines*[2] *an associated set of states* S_{sk}*. It is the case that* $s_0 \in S_{sk}$*.*
- D*, the* decryption algorithm*, is a deterministic algorithm that takes as input a secret key* sk *and a string* s*. If* $s \in S_{sk}$*, then* D *outputs an integer* $i \in \mathbb{Z}_n$*. Otherwise,* D *outputs* $\perp$*.*
- T*, the* transition algorithm*, is a probabilistic algorithm that takes as input the public key* pk*, a string* s*, and an integer* $i \in \mathbb{Z}_n$ *and outputs a string* s'*.*

For any (pk, sk) *output by* $\mathcal{G}(1^k)$*, define* $D' = D(sk, \cdot)$ *and* $T' = T(pk, \cdot, \cdot)$*. Then we require that the set* S_{sk} *along with algorithms* (D', T') *define an* n*-counter. Furthermore, we require that* $D'(s_0) = 0$ *(this represents initialization of the counter to 0).*

SECURITY (HONEST-BUT-CURIOUS). We briefly describe the attack scenario before giving the formal definition. Adversary A is given the public key and the initial state s_0. The adversary then outputs[3] a sequence of integers $i_1, \ldots, i_\ell \in \mathbb{Z}_n$. The state is updated accordingly; that is, the transition algorithm T is run ℓ times, generating $s_1, \ldots, s_\ell$. All intermediate states are given to the adversary, who then outputs $x_0, x_1 \in \mathbb{Z}_n$. A bit b is selected at random, and the counter is incremented by x_b to give state s^*. The adversary, given s^*, must then guess the value of b.

Definition 3. *We say that cryptographic* n*-counter* $(\mathcal{G}, D, T)$ *is secure against honest-but-curious adversaries if, for all poly-time adversaries* A*, the following is negligible (in* k*):*

$$\left| \Pr\left[\begin{array}{l} (pk, sk, s_0) \leftarrow \mathcal{G}(1^k) \\ (i_1, \ldots, i_\ell) \leftarrow A(1^k, pk, s_0) \\ s_1 \leftarrow T(pk, s_0, i_1); \ldots; s_\ell \leftarrow T(pk, s_{\ell-1}, i_\ell) \\ (x_0, x_1) \leftarrow A(s_1, \ldots, s_\ell) \\ b \leftarrow \{0,1\} \\ s^* \leftarrow T(pk, s_\ell, x_b) \\ b' \leftarrow A(s^*) \end{array} : b' = b \right] - 1/2 \right| .$$

[2] Note that membership in S_{sk} may not be efficiently decidable when given only pk. We require, however, that membership *is* efficiently decidable, given sk.

[3] These integers may be chosen adaptively, but for simplicity we present the non-adaptive case here. Note that the construction of Section 3.2 achieves security against an adaptive adversary as well.

SECURITY (MALICIOUS). An honest-but-curious adversary is restricted to having the increment operations (which he must distinguish between) performed on a state distributed according to the output of the transition algorithm T. A malicious adversary, in contrast, is allowed to select the state to be incremented freely. In fact, we allow the adversary to select any *string* to be incremented by T; this allows us to deal with the case in which there is no efficient way to determine whether a string s is a valid state (i.e., whether $s \in S_{sk}$).

Definition 4. *We say that cryptographic n-counter $(\mathcal{G}, D, T)$ is secure against malicious adversaries if, for all poly-time adversaries A, the following is negligible (in k):*

$$\left| \Pr\left[\begin{array}{l} (pk, sk, s_0) \leftarrow \mathcal{G}(1^k) \\ (s, x_0, x_1) \leftarrow A(1^k, pk, s_0) \\ b \leftarrow \{0,1\} \\ s^* \leftarrow T(pk, s, x_b) \\ b' \leftarrow A(s^*) \end{array} : b' = b \right] - 1/2 \right| .$$

VERIFIABLE COUNTERS. It may sometimes be useful to verify whether transitions were indeed computed correctly. For example, when using a counter for voting, it should be publicly verifiable that each voter acted in a correct manner. We therefore define the notion of a *verifiable cryptographic counter* as follows:

Definition 5. *A verifiable cryptographic n-counter is a tuple $(\mathcal{G}, D, T, V)$ such that:*

- *$(\mathcal{G}, D, T)$ is a cryptographic n-counter.*
- *V, the* verification algorithm*, is a probabilistic algorithm satisfying completeness and soundness for all (pk, sk) output by $\mathcal{G}$, as follows:*
  1. *(Completeness) For all $s \in S_{sk}$, if $s' \leftarrow T(pk, s, i)$ for some $i \in \mathbb{Z}_n$, then:*

     $$V(pk, s, s') = 1.$$

     (Note that V does not require i as input.)
  2. *(Soundness) For all s and all strings s' such that for all i, s' is not in the range of $T(pk, s, i)$, the following probability is negligible (in k):*

     $$\Pr[V(pk, s, s') = 1].$$

RESTRICTED COUNTERS. Definitions 1, 2, and 5 may be modified to allow for the possibility that although the counter can store values in $\mathbb{Z}_n$, update operations are restricted to some subset of $\mathbb{Z}_n$. We call counters with this property *restricted.* An illustrative example is a counter used in a voting scheme. Although the counter needs to be able to store values up to L (the number of voters), it may be required to restrict update operations to the set $\{0, 1\}$ (representing a yes/no vote). Modifications to the definitions are straightforward.

ADDITIVE COUNTERS. The transition algorithms described above take an old state s and an integer i and output a new state s' which represents the old value incremented by i. However, definitions 1 and 2 may be modified such that the transition algorithm takes an old state s and a second state s' and then outputs a new state s'' which represents the old value incremented by the value stored in s'. Such counters are termed *additive*. Note that additive cryptographic n-counters include the case of homomorphic encryption over $\mathbb{Z}_n$; yet, the former are more general since the transition algorithm need not be multiplication. Definitions 3 and 4 can be modified for the case of additive counters in the natural way.

3 Constructing Cryptographic Counters

In Sections 3.1 and 3.2, we describe the construction of a cryptographic n-counter based on any 1-bit additive cryptographic counter. We also discuss the extension to the case of verifiable cryptographic counters. In Section 3.4, using as a particular example the encryption scheme based on quadratic residuosity [16] (see Appendix A), which is homomorphic over $\mathbb{Z}_2$, we give an efficient construction of a verifiable cryptographic n-counter where update operations are restricted to $\{0, 1\}$. This provides a natural foundation for a voting protocol; we discuss this connection further in Section 4.

3.1 Linear Feedback Shift Registers

Before presenting our main result, we provide an introduction to the theory of linear feedback shift registers; a more comprehensive treatment can be found in [20, 19]. Let $r_1, r_2, \ldots \in \{0, 1\}$ be a sequence of elements (called *registers*) satisfying the k-th order linear recurrence relation:

$$r_{j+k} = b_k r_{j+k-1} + \cdots + b_1 r_j, \tag{1}$$

where $b_i \in \{0, 1\}$ (throughout this section, addition is over the field $\mathbb{Z}_2$). The sequence $r_1, r_2, \ldots$ is called a *linear recurring sequence.* Once the terms $r_1, \ldots, r_k$ have been fixed, the rest of the sequence is uniquely determined. Define the j-th state of this sequence to be the vector $(r_j, \ldots, r_{j+k-1})$. Equation (1) defines transitions between these states: given state $s = (r_1, \ldots, r_k)$, the next state $s' = (r'_1, \ldots, r'_k)$ can be computed as follows:

$$r'_i = \begin{cases} r_{i+1} & 1 \leq i < k \\ f(r_1, \ldots, r_k) & i = k \end{cases},$$

where the function f is given by (1) as:

$$f(r_1, \ldots, r_k) = b_k r_k + \cdots + b_1 r_1.$$

This sequence of states defines a linear feedback shift register (LFSR). For the present application, it is important to note that f can be computed using XOR operations only.

Since an LFSR has a finite set of states, the sequence of states eventually repeats. The number of states which appear before the first state repeats (and the sequence begins again) is called the *period.* Clearly, an LFSR with period n can be used to count from 0 to $n-1$: choose an arbitrary initial state giving rise to a sequence of period n, label this initial state "0", and label every succeeding state by one more than the label of its predecessor.

It is possible to associate with every LFSR (whose underlying recurrence relation is given by Equation (1)) the characteristic polynomial $g(x) = x^k - b_k x^{k-1} - \cdots - b_1$. The period of an LFSR is related to the order of its characteristic polynomial. In particular, if the characteristic polynomial of an LFSR is *primitive*[4], then the LFSR has maximum possible period $2^k - 1$ (assuming the initial state of the LFSR is not the zero vector) [20, 19]. Primitive polynomials can be generated efficiently using a probabilistic algorithm [22]. It is thus possible to efficiently construct an LFSR which counts from 0 to $n-1$ using the minimum possible $\lceil \log_2 n \rceil$ registers (each representing a single bit).

Given a state s of an LFSR (and assuming knowledge of the initial state), it is easy to decode the state and determine the number it represents by either counting down from s to the initial state, or counting up from the initial state until state s is reached. This requires time $\mathcal{O}(n)$. This procedure is fast, however, even for large[5] n, since each state transition consists of only simple, *bitwise* manipulations (shifts and XORs). More efficient approaches are mentioned in Section 3.3.

3.2 General Construction of a Cryptographic Counter

Theorem 1. *An additive cryptographic* 2*-counter secure against honest-but-curious (resp. malicious) adversaries implies the existence of a cryptographic* n*-counter secure against honest-but-curious (resp. malicious) adversaries, for all* n *of the form* $n = 2^x - 1$.

Sketch of Proof An encryption scheme homomorphic over (the additive group) $\mathbb{Z}_2$ is an example of an additive cryptographic 2-counter secure against honest-but-curious adversaries. For ease of exposition, we describe the construction of a cryptographic n-counter using an encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ which is homomorphic over $\mathbb{Z}_2$; it should be clear, however, that a substantially-similar construction yields a cryptographic n-counter starting from *any* additive cryptographic 2-counter.

We show how to use the encryption scheme as a building block to construct a cryptographic n-counter. First, note that an LFSR (as described in Section 3.1) is an n-counter. The idea behind the construction is as follows: since only XOR operations are needed to effect transitions, the encryption scheme allows

[4] A polynomial $g \in \mathbb{Z}_2[x]$ of degree k is primitive if the smallest integer N for which $g|(x^N - 1)$ is $N = 2^k - 1$.

[5] For a typical voting scheme, n will be on the order of the number of voters. So, even for the U.S. election, we have n only (roughly) 10^8.

a participant to change the counter without leaking any information about the transition. Below is a complete description of the protocol (here, $\ell = \lceil \log_2 n \rceil$):

Key Generation Algorithm $\mathcal{G}'(1^k)$:

1. Run $\mathcal{G}(1^k)$ to generate public key pk_0 and secret key sk_0.
2. Generate a primitive polynomial $g \in \mathbb{Z}_2[x]$ of degree ℓ using [22].
3. Set $r_1 = \mathcal{E}_{pk_0}(1)$ and $r_2 = \mathcal{E}_{pk_0}(0), \ldots, r_\ell = \mathcal{E}_{pk_0}(0)$.
4. Set $s_0 = (r_1, \ldots, r_\ell)$, $sk = (sk_0, g)$, and $pk = (pk_0, g)$. Output pk, sk, and s_0.

Transition Algorithm [defined for $i \in \mathbb{Z}_n$] $T((pk_0, g), (r_1, \ldots, r_\ell), i)$:

1. Polynomial g defines (nonzero) $f(r_1, \ldots, r_\ell) = b_\ell r_\ell + \cdots + b_1 r_1$ (see Section 3.1).
2. Repeat the following procedure i times[6]:
   (a) Set $r'_1 = r_2; \ldots; r'_{\ell-1} = r_\ell$.
   (b) Set $r'_\ell = \prod_{i=1}^{\ell} r_i^{b_i}$.
   (c) Set $r_1 = r'_1; \ldots; r_\ell = r'_\ell$.
3. Set $r'_i = r_i \cdot \mathcal{E}_{pk_0}(0)$, for $1 \leq i \leq \ell$. Output $s' = (r'_1, \ldots, r'_\ell)$.

Decryption Algorithm $D(sk = (sk_0, g), s = (r_1, \ldots, r_\ell))$:

1. Let $r^*_i = \mathcal{D}_{sk_0}(r_i)$, for $1 \leq i \leq \ell$.
2. Let $s^* = (r^*_1, \ldots, r^*_k)$
3. Increment the LFSR defined by polynomial g, beginning with initial state $(1, 0, \ldots, 0)$, until reaching state s^*. Let t be the number of transitions made. Output t.

The protocol described above is a cryptographic n-counter secure against an honest-but-curious adversary. To see this, fix n. The size of the LFSR, ℓ, is thus a constant (independent of the security parameter). A simple hybrid argument shows that an adversary cannot distinguish between random representations of any two states of the counter. Therefore, an adversary cannot gain any information about the current value of the counter, nor about transitions made. We leave a formal proof to the full version of the paper.

Note that if we start with a cryptographic 2-counter secure against malicious adversaries, the above construction is also secure against malicious adversaries. When using an arbitrary encryption scheme homomorphic over $\mathbb{Z}_2$, the above construction is secure against malicious adversaries if it can be efficiently determined (given pk) whether a string represents a valid ciphertext[7]; in this case, the transition algorithm must first check whether every register in s represents a valid ciphertext before computing s' (if this is not true, it aborts). □

[6] This algorithm can be made significantly more efficient to run in time polynomial in $\log n$. This is discussed briefly in Section 3.3.

[7] For example, in the case of encryption using quadratic residuosity, it is possible to tell whether a string C is a valid ciphertext by checking that the Jacobi symbol of C is 1.

In order to make the above construction verifiable, only a few changes are needed. First, we include a random string τ in the public key. Additionally, we change the transition algorithm so that after s' has been output, we append a non-interactive zero-knowledge proof (NIZK) [5] using random string τ that the transition from s to s' was valid. The verification algorithm V runs the proof-verification algorithm for the NIZK proof. If the proof verification succeeds, the verification algorithm outputs 1; otherwise, it outputs 0. A verifiable, restricted n-counter can be constructed in a similar way.

3.3 Observations on the Cryptographic Counter Construction

Linear feedback shift registers have an algebraic interpretation: the state of an ℓ-bit LFSR represents an element of $GF^*(2^\ell)$. Incrementing the counter corresponds to multiplication of the state by a generator, g, of the multiplicative group in $GF^*(2^\ell)$. This allows for two important gains in efficiency, which are highlighted below.

First, the counter may be efficiently updated by values larger than 1. In particular, the counter may be incremented by value i in only $\mathcal{O}(\ell^2 \log i)$ steps, as opposed to the $\mathcal{O}(\ell \cdot i)$ steps used in the transition function of Section 3.2.

Next, note that the state of the LFSR can be viewed as an element of the form g^j in $GF^*(2^\ell)$. Therefore, one can use algorithms for solving the discrete logarithm problem to determine the value represented by the state of the LFSR,. In particular, it is relatively straightforward to determine the value of an ℓ-bit LFSR in time $\sqrt{2^\ell}$, and an algorithm due to Coppersmith [7] allows decoding in time $\mathcal{O}(2^{\ell^{1/3}(\log^{2/3}\ell)})$.

3.4 An Efficient Cryptographic Counter

The well-known encryption scheme based on quadratic residuosity [16] (see Appendix A) is homomorphic over $\mathbb{Z}_2$. Application of Theorem 1 (see also footnote 7) shows that the construction outlined there results in a cryptographic counter secure against malicious adversaries when instantiated with this encryption scheme. If we are interested in verifiability, however, the generic construction of Section 3.2 will be impractical unless there exists an efficient NIZK proof that the transition algorithm was executed correctly. In the case of quadratic residuosity, we show that efficient NIZK proofs are possible. Since we are interested in eventual applications to electronic voting, we focus on the case of a restricted counter where transitions are limited to either no change in the counter (a 0 vote) or incrementing the counter by 1 (a 1 vote).

Consider the cryptographic counter protocol of Section 3.2, instantiated with encryption based on quadratic residues. Let N be a Blum integer which is part of the associated public key. The string $s = (r_1, \ldots, r_\ell)$ (with $r_i \in \mathbb{Z}_N^{+1}$) is a cryptographic representation of some state of the LFSR, but this underlying state cannot be determined unless one knows the secret key. However, following a transition to $s' = (r'_1, \ldots, r'_\ell)$, there are two possibilities: either

$$\mathcal{QR}_N(r'_i) = \mathcal{QR}_N(r_i), \text{for } 1 \leq i \leq \ell, \qquad (2)$$

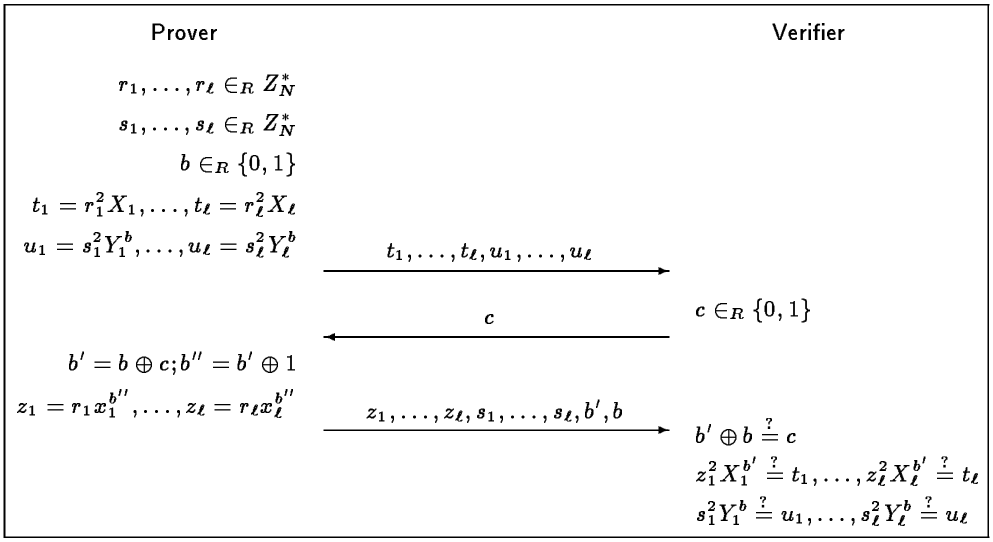
Prover
Verifier
$r_1, \ldots, r_\ell \in_R Z_N^*$
$s_1, \ldots, s_\ell \in_R Z_N^*$
$b \in_R \{0,1\}$
$t_1 = r_1^2 X_1, \ldots, t_\ell = r_\ell^2 X_\ell$
$u_1 = s_1^2 Y_1^b, \ldots, u_\ell = s_\ell^2 Y_\ell^b$
$t_1, \ldots, t_\ell, u_1, \ldots, u_\ell$
$c \in_R \{0,1\}$
c
$b' = b \oplus c; b'' = b' \oplus 1$
$z_1 = r_1 x_1^{b''}, \ldots, z_\ell = r_\ell x_\ell^{b''}$
$z_1, \ldots, z_\ell, s_1, \ldots, s_\ell, b', b$
$b' \oplus b \overset{?}{=} c$
$z_1^2 X_1^{b'} \overset{?}{=} t_1, \ldots, z_\ell^2 X_\ell^{b'} \overset{?}{=} t_\ell$
$s_1^2 Y_1^b \overset{?}{=} u_1, \ldots, s_\ell^2 Y_\ell^b \overset{?}{=} u_\ell$

Fig. 1. Proof of validity for a counter transition.

which represents a 0 vote, or

$$\mathcal{QR}_N(r_i') = \mathcal{QR}_N(r_{i+1}), \text{for } 1 \le i < \ell \text{ and } \mathcal{QR}_N(r_\ell') = \mathcal{QR}_N(\prod_{i=1}^{\ell} r_i^{b_i}), \qquad (3)$$

(with b_i as defined in Section 3.2), which represents a 1 vote. We seek an NIZK proof that either condition (2) or condition (3) holds. Note that these conditions are equivalent to the following: either

$$\mathcal{QR}_N(r_i' \cdot r_i) = 0, \text{for } 1 \le i \le \ell, \qquad (4)$$

or else

$$\mathcal{QR}_N(r_i' \cdot r_{i+1}) = 0, \text{for } 1 \le i < \ell \text{ and } \mathcal{QR}_N(r_\ell' \cdot \prod_{i=1}^{\ell} r_i^{b_i}) = 0. \qquad (5)$$

Therefore, an NIZK proof that one of (4) or (5) holds is sufficient.

In Figure 1 we describe a protocol which takes as input two sequences $X_1, \ldots, X_\ell$ and $Y_1, \ldots, Y_\ell$, and proves the following statement:

$$((\mathcal{QR}_N(X_1) = 0) \wedge \cdots \wedge (\mathcal{QR}_N(X_\ell) = 0)) \vee ((\mathcal{QR}_N(Y_1) = 0) \wedge \cdots \wedge (\mathcal{QR}_N(Y_\ell) = 0)). \qquad (6)$$

By the arguments of the previous paragraph, this is sufficient for our application. The prover knows the square roots of every element of at least one of these sequences[8] (for someone who honestly increments the counter by either 0 or 1,

[8] Without loss of generality, we assume the prover knows the square roots for the first input sequence; thus, in Figure 1, we assume the prover knows $\{x_i\}$ such that $x_i^2 = X_i$, for $1 \le i \le \ell$.

this will be the case); these are the witnesses that these elements are quadratic residues.

By repeating this protocol k_2 times, the probability of cheating is reduced to 2^{-k_2}. This protocol can be made non-interactive using the Fiat-Shamir heuristic [13], by which the challenge of the verifier is replaced by applying a hash function (viewed as a random oracle [1]) to the statement to be proved and the first message of the prover. Let $\mathcal{H}$ be a suitable hash function. The prover need only send $z_1, s_1, \ldots, z_\ell, s_\ell, b', b$ as his proof. The verifier can compute $t_i = z_i^2 X_i^{b'}$ and $u_i = s_i^2 Y_i^b$ and then verify whether $b' \oplus b = \mathcal{H}(X_1, Y_1, t_1, u_1, \ldots, X_\ell, Y_\ell, t_\ell, u_\ell)$.

Theorem 2. *Take the cryptographic counter as described in Theorem 1, instantiated with encryption based on quadratic residuosity. An update of the counter now includes a non-interactive proof (as outlined in Figure 1 and using the Fiat-Shamir heuristic) for statement (6). This then constitutes a verifiable, restricted cryptographic n-counter (for all n of the form $n = 2^x - 1$) which is secure against malicious adversaries.*

Sketch of Proof The protocol given in Figure 1 constitutes an honest-verifier perfect zero knowledge proof with soundness probability 1/2. The proof of this fact follows from techniques outlined in [11]; we refer the reader there for discussion and a complete proof. Repeating the proof k_2 times (non-interactively, using the Fiat-Shamir heuristic) reduces the probability of cheating to 2^{-k_2}, and is a non-interactive zero-knowledge proof (in the random oracle model). The counter is thus *restricted* in that updates are limited to adding an integer from $\{0, 1\}$, and *verifiable* in that updates can be publicly verified as being in this range.

The security of the construction against a malicious adversary follows from Theorem 1 and the zero-knowledge properties of the above protocol. □

3.5 Distributed Decryption of the Counter

We mention that robustness with respect to the trusted authorities can be achieved via distributed generation of the secret key along with threshold decryption of the final counter (which can always be achieved via general multi-party techniques [15]). For the particular case when encryption is done using quadratic residuosity, we are able to achieve efficient distributed key generation and threshold decryption [18]. As this is not the focus of this work, we defer a complete discussion until the full version of the paper.

4 Voting with Cryptographic Counters

We briefly discuss the application of cryptographic counters to the problem of electronic voting. The discussion will be kept as general as possible. For efficient implementation, we have outlined above how it is possible to build an efficient scheme using the encryption scheme based on quadratic residuosity.

We follow the model introduced by Benaloh, et al. [6, 3, 4]. The parties participating in the election consist of a set of voters $V_1, \ldots, V_L$ and a set of authorities

$A_1, \ldots, A_M$, which need not be disjoint. We assume that everyone has access to a *bulletin board* to which all voters will post their messages. Messages are authenticated, and the identity of a sender cannot be forged, nor can messages to the bulletin board be tampered with. Messages are listed in order of arrival (or, equivalently, every message includes the time it was sent), and no one can erase anything from the bulletin board once posted. Note that we do not assume any private channels between voters and the authorities. We now give a high-level description of a voting protocol based on a restricted cryptographic counter; this proves the following theorem:

Theorem 3. *A voting scheme satisfying universal verifiability, privacy, and robustness can be efficiently constructed from any (robust) verifiable, restricted cryptographic counter secure against malicious adversaries (where votes are restricted to the set* $\{0, 1\}$*).*

Sketch of Proof We describe the voting protocol assuming the existence of a verifiable, restricted cryptographic n-counter (where votes are restricted to the set $\{0, 1\}$) secure against malicious adversaries. Robustness (with respect to the authorities) follows if the counter itself is robust (as described in Section 3.5).

SYSTEM SETUP. The authorities run the key generation algorithm for the cryptographic n-counter. Here, n is chosen to be equal to the total number of voters (or an upper bound on the number of voters if the exact number is unknown). If robustness is desired, and/or if some voters are also authorities, the key generation may be done in a robust manner as outlined in Section 3.5. The public key pk and the initial state s_0 are announced to all voters. The key generation step may be the most expensive part of the entire protocol, but it is only a one-time operation which can be done months before the election takes place.

VOTING. The counter always holds the current vote total. The current counter value is always defined as the most recently posted (valid) counter value. Denote the counter after the i^{th} vote by s_i. The $(i+1)^{\text{th}}$ vote is cast as follows: a voter looks at the current counter and computes new state s_{i+1} using the transition function, the previous state s_i, the desired vote $v \in \{0, 1\}$, and the public key pk. The voter publishes this updated state s_{i+1} which then becomes the current state (since it is the most recently posted counter). This proceeds for L rounds until every voter has voted once (see Section 5 for ways to reduce the number of rounds).

Universal verifiability (and hence vote correctness) follows from verifiability of the counter, and voter privacy follows from the definition of security against a malicious adversary. Robustness with respect to the authorities follows from the (robust) distributed key generation and decryption.

TALLYING. When the election is complete, the authorities determine the final tally by decrypting the last (valid) counter. If there is more than one trusted authority, threshold decryption (see Section 3.5) will be necessary. It may also be desirable to have the authorities prove correctness of the decryption; note that it is not acceptable to just publish the secret key, since this would allow determination of every voter's vote retroactively. In the particular case where encryption is done via quadratic residues, the authorities can easily prove that decryption was done correctly by publishing an x for each encrypted value y such that $y = \pm x^2$. □

5 Conclusion

For small-scale elections, the voting scheme outlined here (when based on the encryption scheme using quadratic residuosity) is efficient enough to be practical (cf. Table 1). The required computation and vote size are quite reasonable. One drawback to this scheme is the number of rounds required for voting to take place. When a single cryptographic counter is used, the number of rounds is equal to the number of voters, L. However, by using k cryptographic counters, assigning each voter to one of k groups, and allowing voting to take place in parallel, the number of rounds can be reduced to L/k. Even in a national election, such an approach may be acceptable; for example, by assigning a set of counters to each voting district.

From a theoretical point of view, the approach outlined in this paper is especially interesting since it was previously unclear whether voting could be done efficiently without using fully-homomorphic encryption.

References

1. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. ACM CCCS 1993.
2. J. Benaloh and D. Tuinstra. Receipt-Free Secret-Ballot Elections. STOC 1994.
3. J. Benaloh and M. Yung. Distributing the Power of a Government to Enhance the Privacy of Voters. PODC 1986.
4. J. Benaloh. Verifiable Secret-Ballot Elections. PhD thesis, Yale University, Department of Computer Science, New Haven, CT, 1987.
5. M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and its Applications. STOC 1988.
6. J. Cohen and M. Fischer. A Robust and Verifiable Cryptographically Secure Election Scheme. FOCS 1985.
7. D. Coppersmith. Fast Evaluation of Logarithms in Fields of Characteristic Two. IEEE Transactions on Information Theory, Vol. 30, pp. 587-594, 1984
8. R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-Authority Secret-Ballot Elections with Linear Work. Eurocrypt 1996.
9. R. Cramer, R. Gennaro, and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. Eurocrypt 1997.
10. I. Damgård and M. Jurik. Efficient Protocols Based on Probabilistic Encryption Using Composite Degree Residue Classes. Manuscript, May 2000.
11. A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung. On Monotone Formula Closure of SZK. FOCS 1994.
12. T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Trans. Info. Theory, 31(4): 469–472, 1985.

13. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solution to Identification and Signature Problems. CRYPTO 1986.
14. O. Goldreich. Secure Multi-Party Computation (Working Draft, Version 1.1). Manuscript, 1998.
15. O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game, or a Completeness Theorem for Protocols with an Honest Majority. STOC '87.
16. S. Goldwasser and S. Micali. Probabilistic Encryption. JCSS 28(2): 270–299, 1984.
17. M. Hirt and K. Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. Eurocrypt 2000.
18. J. Katz and M. Yung. Threshold Cryptosystems with Distributed Prime Factors. Manuscript.
19. R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications (Revised Edition)*, Cambridge University Press, 1994.
20. R. Lidl and G. Pilz. *Applied Abstract Algebra (Second Edition)*, Springer, 1997.
21. P. Pallier. Public-Key Cryptosystems Based on Composite Degree Residue Classes. Eurocrypt 1999.
22. J. Rifa and J. Borrell. Improving the Time Complexity of the Computation of Irreducible and Primitive Polynomials in Finite Fields. Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes, 1991.
23. B. Schoenmakers. A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. CRYPTO 1999.

A The Quadratic Residuosity Assumption

These definitions are standard [16, 11]. We say $y \in \mathbb{Z}_N^*$ is a *quadratic residue* modulo N iff there exists an $x \in \mathbb{Z}_N^*$ such that $y = x^2 \bmod N$; otherwise, y is a *quadratic non-residue* modulo N. Define the predicate $\mathcal{QR}_N(y)$ to be 0 iff y is a quadratic residue modulo N, and 1 otherwise. For p prime, the problem of deciding quadratic residuosity is equivalent to computing the Legendre symbol. In fact, the Legendre symbol of y modulo p is defined by $\mathcal{L}_p(y) = +1$ iff y is a quadratic residue, and -1 otherwise.

Now, let $p, q \equiv 3 \bmod 4$ be primes and let $N = pq$ (such N are known as *Blum integers*). No efficient algorithm is known for deciding quadratic residuosity modulo a Blum integer whose factorization is not known. Some information is given by the *Jacobi symbol*, which extends the Legendre symbol as $\mathcal{J}_N(y) = \mathcal{L}_p(y)\mathcal{L}_q(y)$. Despite the way the Jacobi symbol is defined, it is well-known that it can be computed in polynomial time without knowledge of the factors of N. Application of the Chinese Remainder Theorem shows that if $\mathcal{J}_N(y) = -1$, then y cannot be a quadratic residue modulo N. On the other hand, if $\mathcal{J}_N(y) = +1$, no polynomial-time algorithm is known for computing $\mathcal{QR}_N(y)$ if the factorization of N is unknown.

Define $\mathbb{Z}_N^{+1}$ as the set of elements of $\mathbb{Z}_N^*$ with Jacobi symbol 1. It is easy to generate a random $y \in \mathbb{Z}_N^{+1}$ which is a quadratic residue: choose random $r \in \mathbb{Z}_N^*$ and set $y = r^2 \bmod N$. It is equally easy to generate a random quadratic non-residue: choose random $r \in \mathbb{Z}_N^*$ and set $y = -r^2 \bmod N$. This suggests the following semantically secure encryption scheme [16]: the public key is a Blum integer N, and the secret key is the prime factors of N. To encrypt a 0, send a random quadratic residue; to encrypt a 1, send a random quadratic non-residue. This can be extended to n-bit messages in the obvious way, by concatenating n single-bit encryptions.

When $y_1, y_2 \in \mathbb{Z}_N^{+1}$, it is easily verified that $\mathcal{QR}_N(y_1 y_2) = \mathcal{QR}_N(y_1) \oplus \mathcal{QR}_N(y_2)$. This shows that the above encryption scheme is homomorphic over addition in its message space $\mathbb{Z}_2$.