A Practical Scheme for Non-interactive Verifiable Secret Sharing

Paul Feldman

Massachusetts Institute of Technology

Abstract: This paper presents an extremely efficient, non-interactive protocol for verifiable secret sharing. Verifiable secret sharing (VSS) is a way of bequeathing information to a set of processors such that a quorum of processors is needed to access the information. VSS is a fundamental tool of cryptography and distributed computing. Seemingly difficult problems such as secret bidding, fair voting, leader election, and flipping a fair coin have simple one-round reductions to VSS. There is a constant-round reduction from Byzantine Agreement to non-interactive VSS. Non-interactive VSS provides asynchronous networks with a constant-round simulation of simultaneous broadcast networks whenever even a bare majority of processors are good. VSS is constantly repeated in the simulation of fault-free protocols by faulty systems. As verifiable secret sharing is a bottleneck for so many results, it is essential to find efficient solutions.

1. Introduction

1.1 The Problem

Informally, verifiable secret sharing is a protocol in which a distinguished processor, or *dealer*, selects and encrypts a "secret message", s, and gives a "share" of s to each of n processors. There exist parameters t,u such that no t processors can recover s, but any set of u processors are guaranteed that they can easily compute s. When $u=t+1$, we say t is a *threshhold*. The efficiency of a VSS protocol is measured by other parameters as well:

1. The number of rounds of communication required.
2. The number of bits which must be communicated between processors.
3. The number of computations the processors must do.

Another important characteristic is the way in which the processors are guaranteed that they can recover the secret from their shares. All previous schemes have been *interactive*; the validity of a share is proven by an interactive protocol. Here we introduce the concept of a non-interactive VSS, in which a share "proves its own validity". This widens the applicability of VSS to scenarios in which interaction is infeasible, such as sharing a secret among an entire nation. Also, several executions of non-interactive protocols may be run in parallel. By contrast, interactive schemes may have to be run serially. Thus, non-interaction allows us to use VSS as a subroutine without increasing the round complexity.

1.2 History of the Problem

Chor, Goldwasser, Micali, and Awerbuch [CGMA] introduced the notion of VSS. They present a constant round interactive scheme for verifiable secret sharing based on the assumed intractability of factorization. In their solution, $t=O(logn)$, $u=O(n)$; the communication complexity is exponential in t.

The powerful zero-knowledge proof system of Goldreich, Micali and Wigderson [GMW] can be used to create a constant round interactive verifiable secret sharing protocol for any threshhold t. Their solution may be based on the existence of any one-way function.

Benaloh [Be] assumes a reliable public "beacon", and uses it to demonstrate a verifiable secret sharing for any threshhold t running in a constant number of rounds. The beacon may be replaced by an interactive verification. This VSS assumes the existence of hard-to-invert encryption functions with certain properties.

Our contribution is the first non-interactive VSS protocol. Our protocol measures favorably on all of the above parameters. The protocol works for any threshhold t and requires 2 rounds of communication. The communication and computation complexity are small, $O(nk)$ and $O((n\log n+k)(nk\log k))$ respectively, where k is a security parameter (we assume unit cost for broadcasts). We assume the existence of hard-to-invert encryption functions with certain properties; we show that discrete log encryption in either finite fields or on elliptic curves, encryption based on r-th residues, and RSA all have the required properties.

0272-5428/87/0000/0427$01.00 © 1987 IEEE

2. Preliminaries

2.1 The Network

We consider a network of n processors with identities $1,2,...,n$. Each processor, or *player*, is a probabilistic polynomial-time algorithm (PPTA). We assume that every processor has a *broadcast channel*; a message sent on such a channel is received by all processors. Additionally, we assume that there is a private channel from each processor to every other processor. We consider a *semi-synchronous* network. Messages sent at the r-th pulse are received by the $r+1$-st pulse. The period between the r-th and $r+1$st pulses is called the *r-th round*. We shall see, in Section 7, that the assumptions of the broadcast channels and private channels may be relaxed; a complete network is sufficient. A processor may initiate a protocol **P** with *common input* x by broadcasting $\mathbf{P},x$. A protocol is *non-interactive* if all messages are sent by one processor (the *leader*).

A processor is considered *good* as long as he has followed the protocol, and *faulty* once he has deviated from the protocol. The most general (and difficult to guard against) faulty behavior occurs when an adversary coordinates the faulty processors.

Definition: A *(static) t-adversary acting on* **P** is a PPTA A, which need not be one of the n processors, such that

1. A can immediately read any message sent on a non-private channel.

2. At pulse 0, A takes as input the common input x and outputs a t-tuple of processor identities, $(a_1,...,a_t)$, which are immediately *corrupted*. When i is corrupted, his current state and the contents of his tapes become inputs of A. A can replace i's finite state control with any other finite state control. Without loss of generality, A sends messages to i, which i copies instantaneously onto its output tapes.

As a PPTA, A has an output tape; this enables us to formalize the concept that A must "know" something by saying that A outputs it.

We say that A is a *dynamic t-adversary* if A may corrupt as many as t processors in the network at any time. When A corrupts a processor i during **P**, A receives as input only i's current (and future) state(s) and tapes.

2.2 Polynomial Time and the Security Parameter

All cryptographic protocols must assume bounds on the computational power of the players. A parameter of the protocol is a *security parameter* k. Informally, any particular adversary has a *good* chance of "defeating" the protocol only for sufficiently small values of k.

We assume the existence of polynomials $Q_0=Q_0(n,k)$, $Q_1=Q_1(n,k)$ such that all processors can execute $Q_0(n,k)$ steps between pulses, and no processor, or the adversary, can execute $Q_1(n,k)$ steps between pulses. An algorithm is *polynomial-time* if there exists a polynomial Q such that $Q(s)$ is an upper bound on its running time on inputs of length s. Whenever we say that k is an input to an algorithm, we refer to the k-bit string 111...1.

2.3 Mathematical Notation

For a *language* L, L_k consists of all k-bit strings in L. For a string a, $|a|$ is the number of bits in a. Implicit in the notation $\{a,b,...\}\subset S$ is the fact that $a,b,...$ are distinct members of S.

A function U is a *probability distribution* if it assigns to each $Y\in\{0,1\}^*$ a non-negative value $U(Y)$ such that $\sum U(Y)=1$.

A *poly-size family of circuits* is a family $\mathbf{C}=\{\bigcup_{k\in Z} C_k\}$ of probabilistic circuits such that for some polynomial Q, C_k has at most $Q(k)$ inputs and gates.

To emphasize that an algorithm A receives one input we write $A(\cdot)$; if it receives two inputs we write $A(\cdot,\cdot)$ and so on. If U is a probability distribution, then $Y\leftarrow U$ denotes the algorithm which assigns to Y an element randomly selected according to U; that is, Y is assigned the value X with probability $U(X)$. If S is a finite set, then $(y_1,...,y_d)\leftarrow S$ assigns to $(y_1,...,y_d)$ a d-tuple of elements of S with uniform probability. We let $Pr[J(X,Y,...): X\leftarrow S; Y\leftarrow T(X);...]$ denote the probability that the predicate $J(X,Y,...)$ will be true, after the ordered (left to right) execution of $X\leftarrow S$, $Y\leftarrow T(X)$, etc.

2.4 Indistinguishability of Probability Distributions and Zero-Knowledge

Goldwasser, Micali, and Rackoff [GMR] define the notion of *computational indistinguishability*; we shall adapt their definition to our needs.

Let $\mathbf{U}=\{\bigcup_{x\in L} U_x\}$ and $\mathbf{V}=\{\bigcup_{x\in L} V_x\}$ be families of probability distributions. Let $\mathbf{C}$ be a poly-size family of circuits, and let $P(\mathbf{C},\mathbf{U},x)=Pr[b=0: Y\leftarrow U_x; b\leftarrow C_{|x|}(Y)]$. Intuitively, $\mathbf{U}$ and $\mathbf{V}$ are indistinguishable if, for large x, $C_{|x|}$ cannot distinguish the output of U_x from the output of V_x.

Definition: Two families of probability distributions $\mathbf{U}$ and $\mathbf{V}$ over a language L are *indistinguishable* if for any poly-size family of circuits $\mathbf{C}$, $\forall c>0$, $\exists k_0\cdot\forall k\geq k_0\Rightarrow Pr[|P(\mathbf{C},\mathbf{U},x)-P(\mathbf{C},\mathbf{V},x)|>k^{-c}: x\leftarrow L_k]<k^{-c}$.

This notion had already been used by Goldwasser and Micali [GM] in the context of encryption and by Yao [Y] in the context of pseudo-random number generation.

Intuitively, a protocol is zero-knowledge if for any dynamic adversary A acting on it, there is a PPTA which could output strings indistinguishable from those output by A. If all processors have initially blank tapes, then all protocols are zero-knowledge by this definition, since one can construct a PPTA simulating the entire network, including the adversary. Zero-knowledge becomes meaningful when we allow processors to start with private *auxiliary inputs* which are not easily computable functions of the *common input*. For example, assuming *NP* is not contained in *BPP*, one processor may start with a satisfying assignment of a SAT formula, where the formula is given as input to the network.

We define zero-knowledge for a non-interactive protocol **P** in which all processors except the leader start with blank tapes. The leader, i, runs a PPTA *Start* on input (k,n), where k is the security parameter and n is the size of the network. The outputs of *Start* are the common input to **P**, x, and an auxiliary input for i. We assume that A may not corrupt i. Let A output strings according to probability distribution U_x when **P** is initiated on input x.

Definition: A non-interactive protocol **P** is *t-zero-knowledge* if for every dynamic t–adversary A, there exists a PPTA A' which takes as input $x \in L$ and outputs strings according to V_x such that the families of probability distributions $\{U_x\}$ and $\{V_x\}$ are indistinguishable.

3. Verifiable Secret Sharing

We begin by describing ordinary secret sharing in a framework which generalizes naturally to VSS.

3.1. Ordinary Secret Sharing

Ordinary secret sharing enables a dealer to split information among a network so that a quorum of processors is needed to recover the information. An (n,t,u) secret sharing is a pair of PPTAs $(Share(\cdot,\cdot), Recover(\cdots))$; *Recover* takes $u+1$ inputs and is deterministic. The first input of both *Share* and *Recover* is $x \in L$ for some language L. We call $MES_x = \text{Domain}(Share(x,\cdot))$ the *message space*; $|MES_x| > 1$. $CYPH_x = \text{Range}(Share(x,\cdot))$ is the *cypher-text space*. We consider a particular $x \in L$ and omit the argument x. The input of *Share* is a *secret* $w \in MES$, and *Share* outputs an ordered n-tuple $(d_1,\ldots,d_n) \in CYPH$. Each d_i is called a *piece*, or *share*, of w. The input of *Recover* is a u-tuple of ordered pairs $((a_1,c_1),\ldots,(a_u,c_u))$, where $a_i \in [1,n]$, $c_i \in CYPH$, and the a_i are distinct. The output is an element of *MES*. When the input to *Recover* is any labelled subset of u pieces output by *Share* on input $w \in MES$, *Recover* outputs w. Formally,

$$(d_1,\ldots,d_n) \leftarrow Share(w) \Rightarrow \forall \{a_1,\ldots,a_u\} \subset [1,n], \quad (2.1)$$
$$Recover((a_1,d_{a_1}),\ldots,(a_u,d_{a_u})) = w.$$

Finally, we require that w be hard to compute given only t pieces output by *Share*; no static adversary should be able to guess w significantly better than randomly given t labelled pieces.

Definition: We say $(Share(\cdot,\cdot), Recover(\cdots))$ is an (n,t,u) secret sharing if for a language L,

1. $\forall x \in L \ \forall w \in MES_x, (d_1,\ldots,d_n) \leftarrow Share(x,w) \Rightarrow$
$\forall \{a_1,\ldots,a_u\} \subset [1,n], Recover(x,(a_1,d_{a_1}),\ldots,(a_u,d_{a_u})) = w.$

2. $\forall$ PPTAs $A(\cdot)$, $Guess(\cdots), \forall c > 0, \exists k_0, |x| \geq k_0 \Rightarrow$
$\Pr[w = Guess(x,(a_1,d_{a_1}),\ldots,(a_t,d_{a_t})): (a_1,\ldots,a_t) \leftarrow A(x);$
$w \leftarrow MES_x; (d_1,\ldots,d_n) \leftarrow Share(x,w)] < 1/|MES_x| + |x|^{-c}.$

Later, we shall strengthen property 2, by allowing a dynamic adversary to choose which pieces of the secret to take as input.

Shamir [S] presents an $(n,t,t+1)$ secret sharing for any threshhold t. Let L be the set of prime numbers greater than n. For $p \in L$, we define $Share = Share(p,\cdot)$ as follows. The domain is $MES = Z_p$. Protocol *Share*, on input $w \in MES$, sets $y_0 = w$ and lets $(y_1,\ldots,y_t) \leftarrow Z_p$. Let $Q(s)$ be the polynomial $\sum_{i=0}^{t} y_i s^i$. Then the output of *Share* is $Q(1),Q(2),\ldots,Q(n) \bmod p$. *Recover* is polynomial interpolation, which is used to find $Q(0) = w$. Informally, we argue that t pieces are no help in recovering w. Given the value of Q at any t points, then the value at 0, namely $y_0 = w$, uniquely specifies a polynomial Q. Since all other coefficients were chosen uniformly, the chance that a particular polynomial was picked is directly proportional to the probability that its constant term was the secret chosen. Therefore, seeing t pieces gives no additional information.

An (n,t,u) secret sharing $(Share, Recover)$ suggests a non-interactive protocol in which the leader, or *dealer*, can "split" a secret among n players in such a way that only a quorum of u can recover the secret. Namely, the dealer broadcasts x and sends piece d_i to player i via private channel. When u players wish to recover the secret, each broadcasts his labeled piece, and *Recover* can be run to return the secret.

Can this protocol be used if t players may be faulty? No! It is true that no set of t faulty players can recover the secret themselves. However, even if $u < n - t$, in which case there are enough good players to recover the secret, bad players may interfere. For example, a bad player i may broadcast a spurious value in place of d_i. Good players cannot distinguish good pieces from spurious pieces. If $t = \theta(n)$, then a random u-tuple consists entirely of good players with exponentially small probability.

Indeed, the dealer can prevent this problem by authenticating the pieces. In this way, good players could recognize which pieces truly came from the dealer. However, a more endemic problem is the following: what happens if the dealer is faulty? The good players themselves might get authenticated yet spurious pieces, and they will not be able to recover the secret. Depending on the behavior of *Recover* on "invalid" inputs, the good players may not be aware that different sets of pieces would return different secrets. The secret returned could depend on which bad players chose to broadcast their authenticated, spurious pieces. The best kind of secret sharing would allow any player to verify during *Share* whether or not his piece is valid; moreover, he should be able to check during *Recover* whether or not the piece another player broadcast is valid.

3.2 Verifiable Secret Sharing

Informally, a verifiable secret sharing protocol must meet the following two requirements:

1. Verifiablility constraint: upon receiving a share of the secret, a player must be able to test whether or not it is a valid piece. If a piece is valid, there exists a unique secret which will be output by *Recover* when it is run on any u distinct valid pieces.

2. Unpredictability: there is no polynomial-time strategy for picking t pieces of the secret, such that they can be used to predict the secret with any perceivable advantage.

This framework allows for an interactive protocol proving validity of the pieces. A VSS protocol is non-interactive if there is a polynomial-time algorithm *Check* which tests validity of the pieces. Obviously, the same pieces are not valid for different secrets; we introduce a PPTA *Encrypt* to handle this.

Informal Definition: An *(n,t,u) non-interactive verifiable secret sharing* is a quadruple of PPTAs $(Share, Recover, Check, Encrypt)$ such that

1. $(Share, Recover)$ is an (n,t,u) secret sharing over a language L.

2. $\forall x \in L, \forall w \in MES_x, \forall 1 \le j \le n,\ Y \leftarrow Encrypt(x,w);$
$(d_1,\ldots,d_n) \leftarrow Share(x,w) \Rightarrow,\ Check(x,Y,j,d_i)=1.$

3. $\forall x \in L, \forall Y \in \text{Range}(Encrypt(x,\cdot)),\ \exists w \in MES_x \cdot \ni \cdot$
$\forall \{a_1,\ldots,a_u\} \subset [1,n], \forall d_1,\ldots,d_u \in MES_x,$
$(Check(x,Y,a_i,d_i)=1\ \forall 1 \le i \le u) \Rightarrow$

$$Recover(x,(a_1,d_1),\ldots,(a_u,d_u))=w.$$

Remark: Actually, we require that $(Share, Recover)$ remains an (n,t,u) secret sharing even when $Y \leftarrow Encrypt(x,w)$ is given; *Share* itself may take Y as an input. To formalize the definition, *Share* itself would set $Y \leftarrow Encrypt(x,w)$ and append Y to each piece.

3.3 Applications of Verifiable Secret Sharing

Verifiable secret sharing enables a communication network to simulate a simultaneous broadcast network. Informally, every processor is required to share his round r message before any round r message is revealed. This trivializes the design of protocols such as secret bidding, leader election, and flipping a fair coin.

In the simulation of a simulataneous broadcast network, each processor acts as dealer. For an interactive VSS such as [GMW], Chor and Rabin [CR] show that $\log n$ rounds are sufficient for n processors to prove the validity of their pieces; it is not known whether this can be done in a constant number of rounds. In a non-interactive VSS, all processors may deal secrets in parallel; therefore, our protocol gives constant round solutions to all these problems.

Another application is achieving a fast Byzantine Agreement without any preprocessing. The best previously known algorithm which could tolerate a linear number of faulty processors, due to Chor and Coan [CC],[C], required expected $O(n/\log n)$ rounds. Feldman and Micali [FM] show that running non-interactive verifiable secret sharing without broadcast channels, using Crusader Agreement instead, yields a constant expected time Byzantine Agreement protocol.

Even beyond the protocols which directly follow from it, verifiable secret sharing now plays a central role in cryptographic protocol design in a most dramatic way. In [GMW] it is shown that all protocols can be designed to resist t faulty players out of $2t+1$. Verifiable secret sharing is one of the three main blocks needed for their simulation. This highlights even more the need for an efficient verifiable secret sharing, as it may enter as a key subroutine in a large class of protocols.

4. Motivation for Our Solution

The motivation behind our solution is to utilize homomorphic relationships which may exist between values and their encryptions. For a certain class of encryption schemes, which we shall call *homomorphic*, we can construct an algorithm *Check* which enables a player to verify the validity of his piece.

4.1. Probabilistic Encryption

A problem with deterministic encryption schemes is that it is easy to check whether or not a given ciphertext is the encryption of any given message. Goldwasser and Micali [GM] showed how to overcome this problem by *probabilistic* encryption. They define the notion of a family of *unapproximable predicates*. Let $\mathbf{H}=\{Hard_x : x \in L\}$ be a family of *predicates*, each $Hard_x$ maps $CYPH_x \rightarrow \{0,1\}$. Elements which map to 0 (1) are considered encryptions of 0 (1). Intuitively, such an

encryption is secure if there is no efficient way of computing $Hard_x$ on random elements; **H** is *unapproximable* if no poly-size family of circuits can compute $Hard_x(y)$ significantly better than randomly guessing when x and y are randomly selected.

Let $\mathbf{C}=\{C_k: k\in\mathbf{Z}\}$ be a poly-size family of circuits, $C_k(\cdot,\cdot)$ takes as input $x\in L_k$ and $y\in CYPH_x$ and outputs a bit. For $x\in L_k$, let
$P(\mathbf{C},k,x)=Pr[C_k(x,y)=Hard_x(y): y\leftarrow CYPH_x]$.

Definition: The predicate **H** is *unapproximable* if

$$\forall \mathbf{C}=\{C_k: k\in\mathbf{Z}\},\forall c>0,\ \exists k_0\cdot\forall k\geq k_0\Rightarrow \quad (4.1)$$
$$Pr[P(\mathbf{C},k,x)>.5+k^{-c}: x\leftarrow L_k]<k^{-c}.$$

To probabilistically encrypt a bit using $Hard\in\mathbf{H}$, there must be a way for the encrypter to find "random" elements which map to 0 (or 1). One possibility is if *Hard* is a *trapdoor* function, that is, *Hard* is easy to compute given a short string *Hint*. In this scenario, the encrypter generates a pair $x,Hint_x$; he can then compute $Hard_x$ on random elements of $CYPH_x$, and picks one which maps to the desired bit.

Another method exists if *Hard* is an easy predicate composed with a one-way function, as we now explain. Let *MES* be a message space, and let *Encrypt* be an injection from $MES\rightarrow CYPH$. Informally, *Encrypt* is one-way if *Encrypt* is easy to compute, but $Encrypt^{-1}$ is hard to compute. Let $Predicate: MES\rightarrow \{0,1\}$ be an easily computable function. Suppose $Hard=Predicate(Encrypt^{-1})$. Then the encrypter can randomly pick $y\in CYPH$, and compute $z=Encrypt(y)$ and $b=Predicate(y)$. By construction, $Hard(z)=b$, hence z is a probabilistic encryption of b. All encryptions we shall consider will be computed by this latter method, even though some are also trapdoor schemes.

Example: RSA probabilistic encryption, developed by Rivest, Shamir, and Adleman [RSA] may be computed either way. Let m be a product of large primes, and e a number such that $(e,\phi(m))=1$. Let $d=e^{-1}$ mod $\phi(m)$; d, which is not easy to compute without knowing the factorization of m, is the trapdoor hint. We define $CYPH=Z_m^*$. For $y\in Z_m^*$, we define $Hard(y)$ to be the parity of y^d mod m. This is easy to compute for anyone knowing d, but it is believed to be hard to compute otherwise. We shall focus on the method of encrypting bits without knowing d. Let $Encrypt(y)=y^e$ mod m and $Predicate(y)$ be the parity of y; both are easy to compute. A 0 (1) is encrypted by raising a random even (odd) element of Z_m^* to the e power mod m.

We shall find it convenient to generalize this notion by enlarging the range of *Predicate*, and hence *Hard*, to $[1,l]$; the value of l is a parameter of the particular encryption function.

4.2 Homomorphic Encryption Functions

Often, a rich algebraic structure underlies an encryption scheme. Relations among cleartext values may imply relations among the encryptions. For example, in RSA encryption, $Encrypt(yz)=(yz)^e$ mod $m=Encrypt(y)Encrypt(z)$. More generally, when both the domain and range of *Encrypt* are groups, *Encrypt* may be a homomorphism of the groups. Benaloh [Be] utilized such homomorphisms in his secret sharing, and pointed out that our VSS extends to such a class of encryption functions [Be2]. Let
$MES=\text{Domain}(Encrypt)$ be an additive group, and $CYPH=\text{Range}(Encrypt)$ be a multiplicative group. The key property is that for all $B,C\in MES$,

$$Encrypt(B+C)=Encrypt(B)\cdot Encrypt(C) \quad (4.2)$$

For $c\in\mathbf{Z}$, we define the scalar product $c\cdot B=B+B+...+B$ with c summands. Induction may be used to show that $\forall B\in MES$, $\forall c\in\mathbf{Z}$, $Encrypt(c\cdot B)=(Encrypt(B))^c$.

Security can only be achieved by picking among a family of encryption functions. Let $Generator(\cdot,\cdot)$ be a PPTA which, on input k,n, selects $x\leftarrow(L_k\cap L^n)$ for some language L; we define $L^n\subset L$ below. We impose uniformity constraints by requiring a PPTA $Encrypt(\cdot,\cdot)$ such that $Encrypt(x,\cdot)=Encrypt_x(\cdot)$, and similarly for $Predicate(\cdot,\cdot)$. Likewise, there must be uniform algorithms for computing the group operations, uniformly sampling *MES*, and the following function *Divide*, whose purpose will first become clear in Section 5.3. For all $B\in MES_x$, $Divide(x,n,n!\cdot B)=Predicate(x,B)$. Since scalar multiplication by $n!$ need not be a 1-1 function, the existence of *Divide* imposes a certain structure on *Predicate*. Moreover, this says that $Predicate_x(B)$ is easily computable given only $n!\cdot B$ as input. We define $MES_x^s=\{B\in MES_x: Predicate(B)=s\}$. We define $L^n=\{x\in L: (Divide(x,n,\cdot)$ is well defined) and

$$(\,|\text{Range}(Predicate(x,\cdot)\,|\geq n)\}.$$

If these properties are satisfied, and $Hard=Predicate(Encrypt^{-1})$ is unapproximable, we say *Generator* is a *homomorphic probabilistic encryption scheme generator.* Equation (4.1) implies that *NP* is not contained in *BPP*, so we will need to make certain unproven complexity assumptions to assert that we have such generators.

In Section 8, we construct homomorphic probabilistic encryption scheme generators based on different problems. One is based on the difficulty of taking discrete logs in a finite field; a suitable restriction of the domain is required. The same method lets us base a generator on the difficulty of taking discrete logs on a elliptic curves. Benaloh [Be2] has pointed out that a generator may be based on the difficulty of distinguishing r-th powers in Z_m^*, where r is a prime dividing $\phi(m)$; the nature of these probabilistic encryptions differs slightly from the description given here. RSA is

sufficiently homomorphic for our purposes if we define the "addition" on the domain $MES = Z_m^*$ to be multiplication mod m.

4.3. Using a Homomorphic Probabilistic Encryption Scheme to Produce a Non-interactive (n,t,t+1) VSS

We restrict the rest of this chapter to an informal discussion of our protocol. The formal presentation is given in Section 5.

Given *Encrypt*, we show how to share a secret in $[1,l]$; a longer secret may be shared by sharing blocks of $\lfloor l \rfloor$-bit secrets in parallel. We convert Shamir's secret sharing into a non-interactive VSS as follows. The dealer uniformly picks a secret $s \leftarrow [1,l]$, and $y_0 \in MES^s$ and sets y_0 to be the constant term of a degree t polynomial Q. He chooses the t other coefficients of Q uniformly in *MES*. He then broadcasts encryptions of the coefficients of Q, $Encrypt(y_0),...,Encrypt(y_t)$. As in Shamir's scheme, he sends $Q(j)$ to player j via a private channel. The point is that j can verify his piece by checking that

$$Encrypt(Q(j)) = \tag{4.3}$$
$$(Encrypt(y_0)) \cdot (Encrypt(y_1))^j \cdots (Encrypt(y_t))^{j^t}$$

Even for probabilistic encryption schemes, we must still prove that broadcasting the encryptions does not allow an adversary to guess the secret advantageously. This is done by showing that the adversary can simulate his view by himself. To facilitate the simulation, we alter the protocol slightly by having the dealer encrypt a certain multiple of the coefficients, as will be described below.

4.4 Tolerating a Dynamic Adversary

It would seem that a non-interactive VSS could not be zero-knowledge with respect to a dynamic adversary. On the one hand, by the intractability assumption, a simulator cannot reconstruct the secret. On the other hand, we wish that he can simulate the output of an adversary, who could corrupt **any** t players and output valid pieces with those indices. However, if the simulator knew all the pieces, he could compute the secret, The resolution of this difficulty is to permute the shares in a way unknown to the adversary; the adversary does not know which share a player should have before corrupting him. In this way, a simulator knowing t shares can "fool" an adversary into thinking that these are the correct shares for the players corrupted. Of course, this leaves the problem of how to convince a player that he is receiving the proper share.

This latter problem is solved by having the dealer probabilistically encrypt a random permutation $\pi \in S_n$. The dealer lets $(A_1 \leftarrow MES^{\pi(1)}),...,(A_n \leftarrow MES^{\pi(n)})$ and broadcasts $Encrypt(A_1),...,Encrypt(A_n)$. The dealer sends A_h on the private channel to player j, where $h = \pi^{-1}(j)$; j can verify that $Predicate(A_h) = j$. This designates that player j should subsequently receive $Q(h)$. Player j is not convinced that the dealer encrypted a real permutation, but he knows that the h-th piece is designated exclusively for him, since only A_h can encrypt to $Encrypt(A_h)$ and $Predicate(A_h) = j$. To facilitate the simulation, this step will precede the broadcast of the encrypted variables.

With respect to a static adversary, the simulation may be done without permuting the shares. We argue without proof, in Section 9.2, that the shares need not be permuted even against a dynamic adversary, although there are problems with the simulation in that case.

5. Our Protocol

5.1 Initialization

Let *Generator* be a homomorphic probabilistic encryption scheme generator. The dealer i sets $x \leftarrow Generator(k,n)$, where k is the security parameter, and n is the size of the network. We omit the dependence on x. Let $MES = \text{Domain}(Encrypt)$, $CYPH = \text{Range}(Encrypt)$.

The dealer i uniformly picks a secret $s \leftarrow [1,l]$, sets $y_0 \leftarrow MES^s$, and computes $Y = Encrypt(y_0)$. The VSS is initialized with common input i,x,Y. All players store i,x,Y on a work tape; i additionally stores y_0 as his auxiliary input.

5.2. The Protocol Share

The dealer broadcasts messages in steps 1 and 3; the players perform calculations in steps 2 and 4.

1. The dealer i selects $\pi \leftarrow S_n$, a uniformly chosen permutation of $[1,n]$. He sets $A_j \leftarrow MES^{\pi(j)}$ for each j and broadcasts the probabilistic encryption this specifies, $Encrypt(A_1),...,Encrypt(A_n)$. He sends h, A_h on the private channel to j, where $h = \pi^{-1}(j)$.

2. Each player j lets $\vec{A} = (Encrypt(A_1),...,Encrypt(A_n))$ denote the values he received on i's broadcast channel and h and C denote the values he received on the private channel from i in round 1. We define $Checkid(j,h,C,\vec{A}) = 1$ iff $Encrypt(C) = Encrypt(A_h)$ and $Predicate(C) = j$. When this is the case, j stores h but not $C = A_h$; if this fails, j immediately rejects the dealer as faulty.

3. The dealer sets $(y_1,...,y_t) \leftarrow MES$. He computes and broadcasts $(Y_1,...,Y_t) = (Encrypt(n! \cdot y_1),...,Encrypt(n! \cdot y_t))$. Let $Q: \mathbf{Z} \rightarrow MES$ denote the "polynomial" function, $Q(a) = \sum_{l=0}^{t} a^l \cdot y_l$. The dealer privately sends $M_h = Q(h)$ to player $j = \pi(h)$.

4. Each player j computes $Y_0 = Y^{n!}$. Let $\vec{Y}=(Y_0, Y_1, \ldots, Y_t)$ denote Y_0 and the values broadcast last round; let N_h be the value j just received on the private channel. Define $Check(\vec{Y}, N_h, h)=1$ iff $Encrypt(n! \cdot N_h)=(Y_0)\cdot(Y_1^h)\cdots(Y_t^{h^t})$. When this is true, j stores h, N_h; otherwise, he rejects the dealer as faulty.

Remark: As we mentioned in Section 4.2, scalar multiplication by $n!$ need not be 1-1. Therefore, coefficient y_a is not uniquely determined by the encryption $Y_a = Encrypt(n! \cdot y_a)$. However, *Encrypt* is 1-1, so $n! \cdot y_a$ is uniquely determined by Y_a. Therefore, if $Check(\vec{Y}, N, h)=1$, this only guarantees that $n! \cdot N = n! \cdot Q(h)$. Fortunately, this is good enough, since it suffices to find $n! \cdot y_0$ to recover the secret.

5.3 The Protocol Recover

At any time after *Share* was executed on input (i, x, Y), *Recover* may be initiated on the same input. Each player j retrieves and broadcasts the private values he stored for that execution, h, N_h. Each j runs $Check(\vec{Y}, N_l, l)$ to test whether an indexed piece which was broadcast, l, N_l, is valid. Assume that j can find $t+1$ valid indexed pieces $((a_1, N_{a_1}), \ldots, (a_{t+1}, N_{a_{t+1}}))$. The polynomial Q is found by interpolation, $Q = \sum_{l=1}^{t+1} \left(\prod_{h \neq l} \frac{(x - a_h)}{(a_l - a_h)} \right) \cdot N_{a_l}$. We observe that this formula would give a polynomial-time algorithm for computing Q if we could perform scalar division. In particular, $y_0 = Q(0) = \sum_{l=1}^{t+1} \left(\prod_{h \neq l} \frac{(a_h)}{(a_h - a_l)} \right) \cdot N_{a_l}$. Although we cannot necessarily do arbitrary scalar division, the denominator of each term of the sum divides $n!$, so $n! \cdot y_0$ may be computed by additions and scalar multiplications by integers. Therefore, j can compute $n! \cdot y_0$ and hence $Divide(n, n! \cdot y_0) = Predicate(y_0) = s$.

5.4 An Extension of Share

In [F], we modify *Share* by allowing each player to broadcast a single termination message. On the basis of these messages, the players can determine whether or not enough good players received good pieces to ensure recovery of the secret. This determination is based solely on broadcast messages, hence all good players reach the same conclusion.

6. A Proof of the Security

6.1. The Zero-Knowledge Simulation

Let A be any dynamic t– adversary. We construct a PPTA A' which simulates the whole network, including A, without benefit of the auxiliary input. Assume that *Share* is initiated with common input i, x, Y; henceforth, we omit the argument x.

1. Since i does not utilize the auxiliary input, namely y_0, in round 1, A' simulates i (and A) perfectly at step 1.

2. Likewise, A' perfectly simulates A and all other players in step 2.

3. Let $j_1, \ldots, j_{t_0}$ denote the players A corrupts by the end of step 2, and let $\pi(h_l) = j_l$ for $l \in [1, t_0]$. A' picks $h_{t_0+1}, \ldots, h_t$ uniformly among the as-yet unselected indices in $[1, n]$, so $h_1, \ldots, h_t$ are all distinct. Let $h_0 = 0$, and let $H = \begin{pmatrix} 1 & h_0 & \cdots & (h_0)^t \\ 1 & h_1 & \cdots & (h_1)^t \\ \vdots & \vdots & \cdots & \vdots \\ 1 & h_t & \cdots & (h_t)^t \end{pmatrix}$. This is a Vandermonde matrix, and its determinant is $D = \prod_{0 \leq l < e \leq t} (h_e - h_l)$.

Lemma 6.1: Let $H = \{h_{le}\}$, where l and e range from 0 to t and $h_{le} = h_l^e$; let $G = H^{-1} = \{G_{le}\}$. Then the denominator of G_{le} divides $\prod_{j \neq e} (h_j - h_e)$.

This is proved by showing that $\prod_{j \neq e} \prod_{l \notin \{j, e\}} (h_j - h_l)$ divides the determinant of the l, e minor of H. Since $\{h_0, \ldots, h_t\} \subset [0, n]$, the denominator of G_{le} divides $n!$ for any e.

A' sets $(M_1, \ldots, M_t) \leftarrow MES$. We implicitly set $M_0 = y_0$ (which is well defined as $E^{-1}(Y)$).

We define the $*$-inner product by $(c_1, \ldots, c_d) * (B_1, \ldots, B_d) = \sum_{i=1}^{d} c_i \cdot B_i$, for $c_1, \ldots, c_d \in \mathbf{Z}$ and $B_1, \ldots, B_d \in MES$. This extends to a $*$-matrix product in the natural way. Observe that the conditions $M_{h_l} = Q(h_l)$, for $l \in [0, t]$ may be written as a matrix equation $\vec{M} = H * \vec{y}$, where $\vec{M} = \begin{pmatrix} M_0 \\ M_1 \\ \vdots \\ M_t \end{pmatrix}$ and $\vec{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_t \end{pmatrix}$ are considered as column vectors. By $*$-multiplying both sides by $G = H^{-1}$, we obtain $G * \vec{M} = G * (H * \vec{y}) = (G \cdot H) * \vec{y} = \vec{y}$ (the associative law follows from direct calculation). Denote row l of $(n!) \cdot G$ by the integers $g_{l0}, \ldots, g_{lt}$, so $\sum_{e=0}^{t} g_{le} \cdot M_e = n! \cdot y_l$. Applying *Encrypt* to both sides and using equation 4.2,

$$\prod_{e=0}^{t} (Encrypt(M_e))^{g_{le}} = Encrypt(n! \cdot y_l) \qquad (6.1)$$

Observe that A' can compute the left hand side, since $Encrypt(M_0) = Y$ is part of the common input and M_e was chosen by A' for $e > 0$. The right hand side is the definition of Y_l. Therefore, at step 3, A' broadcasts $(Y_1, \ldots, Y_t)$ and sends M_l to j_l for $1 \leq l \leq t_0$. By construction, $Check(\vec{Y}, M_l, j_l)=1$.

Let j_{t_0+1} be the next player A corrupts; A' simulates j_{t_0+1} starting from the end of round 2, with $i, x, Y, h_{t_0+1}, \vec{A}$ as the only values on j_{t_0+1}'s work tape. By construction, A' is able to give the proper piece M_{t_0+1} at round 3. In like manner, the next player corrupted will have stored the value h_{t_0+2}, and so on.

4. A' simulates corrupted players by running A ansd outputs the output of A.

6.2 Indistinguishability of the Output of A from the Output of A'

Actually, *Share* is zero-knowledge even if *Encrypt* is easy to invert, in which case i's auxiliary input is easily computable, and A' can perfectly simulate the entire network and output what A outputs. For simplicity, we shall prove that *Share* is zero-knowledge assuming equation 4.1, which implies that *Encrypt* is hard to invert. If the output of A is distinguishable from the output of A', then A distinguishes the probability distributions on its inputs. Thus, it suffices to show that the inputs to A acting on *Share* are indistinguishable from the inputs to A in the simulation.

The selection of $\pi, A_1, \ldots, A_n$ is done exactly as in *Share*. Also, the probability distribution on $\overline{y}=y_1,\ldots,y_t$, which is the uniform one when A acts on *Share*, is also uniform in the simulation for the following reason. A' selected all elements of $\overline{M}=M_1,\ldots,M_t$ uniformly in *MES*. Given fixed values of M_0 and y_0, there is a 1-1 onto map between choices of $\overline{M}$ and choices of $\overline{y}$ -- H sends $\overline{M}=(M_0,\overline{M})$ to $\overline{y}=(y_0,\overline{y})$, so the uniform distribution on one implies the uniform distribution on the other. In *Share*, the elements of $\overline{y}=y_1,\ldots,y_t$ were selected uniformly. Since in both cases, $\overline{M}$ is uniquely determined by $\overline{y}$, this also has identical distribution.

There is only one difference in the inputs to A in the two cases. When A acts on *Share*, if player j is corrupted after round 2, j stored h where $h=Predicate(A_j)=Hard(Encrypt(A_j))$. In the simulation, the l-th player j corrupted by A after round 2 will have stored the value h_{t_0+l}, regardless of the actual value $Hard(Encrypt(A_j))$. Therefore, any PPTA which can distinguish the views can determine $Hard(Encrypt(A_j))$ better than randomly guessing, given only $Encrypt(A_j)$. However, by (4.1), A cannot guess $Hard(E(A_j))$ better than randomly. Therefore, the inputs to A are indistinguishable.

7. Extending the Protocol to a Larger Class of Networks

We informally remark that *Share*, *Recover* may be implemented without private channels and/or broadcast channels; a complete network is sufficient. Formal proof of the claims in this section will appear in [F].

We first observe that *random pads* can be used to simulate private channels. Assume that the dealer once privately sent a long string of random bits, r_j, to each player j, but all future messages may be input by an adversary. To privately send message m_j to j, he sends $m_j \oplus r_j$, a bitwise exclusive-or of the real message and an "unused" portion of the random pad. Player j, who knows r_j, easily computes m_j. The adversary, who does not know r_j cannot distinguish the sent message from random noise.

A random pad r_j may be sent privately by the use of trapdoor encryption functions. Suppose that player j broadcasts an encryption function, $Encrypt_j$ for which he knows a trapdoor hint, $Hint_j$. The dealer picks a random pad r_j and computes a probabilistic encryption of it, $Encrypt_j(r_j)$ and sends this to j. Player j decrypts to find the pad r_j. Assuming $Encrypt_j$ is secure, an adversary, on input $Encrypt_j(r_j)$, cannot guess any bit of r_j significantly better than at random.

Although our algorithm only required that the dealer send messages along a private channel, this technique can be used to simulate, simultaneously, a complete network of private channels. In particular, all players may still "deal" secrets simultaneously.

The assumption of a broadcast channel is very strong. We can eliminate this assumption by simulating the broadcast channel by Crusader Agreement, which is a weak form of Byzantine Agreement. The simplest version requires the assumption that $t<n/3$; a more involved version tolerates any value of $t<n/2$, provided that the processors start with secure signature schemes.

8. Candidates for Homomorphic Encryption Schemes

8.1 Encryption Based on Discrete Logs in Finite Fields

Generator, on input (k,n) uniformly selects a k-bit prime p and the prime factorization of $p-1$ subject to the condition below. (Bach [Ba] shows how to uniformly generate numbers of known factorization.) Let $d_{r,n}=\prod_{\substack{q\le n \\ q \text{ prime}}} (q^r, r)$, that is, the largest divisor of r without a prime factor exceeding n. We require that $d_{p-1,n}<n^3$. For random k-bit numbers r, $d_{r,n}$ approximates a normal distribution with expected value n ', so $d_{p-1,n}<n^3$ for all but an exponentially small fraction of p, and this remains true when we restrict p to k-bit primes. (We could use any polynomial in place of n^3). Since the only fast algorithms for taking discrete logs require that $p-1$ has only small prime factors, any p which did not satisfy this condition would not be considered a good choice for discrete log encryption in any case.

Generator uniformly selects a generator of the multiplicative group Z_p^*, g. The index x is the concatenation p,g; we omit reference to it. The domain of *Encrypt* is Z_{p-1}, and the range is Z_p^*. For $y\in Z_{p-1}$, $Encrypt(y)=g^y \bmod p$. This satisfies the homomorphism property, $Encrypt(B+C)=g^{B+C} \bmod p = Encrypt(B)\cdot Encrypt(C)$. It is easy to design PPTAs for the group operations and sampling *MES*; we describe *Predicate* and *Divide* below. We make the following intractability assumption for taking discrete logs:

$$\forall c>0, \forall n>0, \forall \text{PPTA } Logarithm(\cdot,\cdot,\cdot), \quad (8.1)$$
$$\exists k_0 \cdot \forall k \ge k_0 \Rightarrow Pr[y=Logarithm(p,g,z):$$
$$(p,g)\leftarrow Generator(k,n); y\leftarrow Z_{p-1}; z=g^y \bmod p] < k^{-c}$$

Notice that $n!$, being even, does not have a multiplicative inverse in $MES = Z_{p-1}$. A necessary condition for the existence of *Divide* is that

$\forall B, C \in MES, n! \cdot B = n! \cdot C \Rightarrow Predicate(B) = Predicate(C)$.

Let $e = (p-1)/d_{p-1,n}$. We partition *MES* into equivalence classes mod e, so $B \equiv C \bmod e \Rightarrow Predicate(B) = Predicate(C)$. By construction, $(e, n!) = 1$, so $n! \cdot (B-C) \equiv 0 \bmod p-1 \Rightarrow B \equiv C \bmod e$. Division by $n!$ is easy mod e, so *Divide* is no harder to compute than *Predicate*.

Long and Wigderson [LW],[L] show the equivalence of computing discrete logs and guessing whether or not the log is in the top half of the residues mod $p-1$. In fact, the ability to guess any predicate of the top $O(log\ |p\ |)$ bits of the log better than at random is equivalent to computing the log. Kaliski [Ka] generalizes this result to apply to logarithms in arbitrary commutative groups. There is a correlation between the most significant bits of $f \leftarrow Z_{p-1}$ and the most significant bits of f mod e, hence the ability to predict one better than guessing implies the same for the other.

Set l, the size of *MES*, to be $2^{|k|}$. For $f \in Z_e$, define $Predicate(f) = [(f \bmod e) l/e]$. Then equation 8.1 implies equation 4.1, that *Hard* is hard to compute.

8.2 Discrete Logs on Elliptic Curves

A homomorphic probabilistic encryption scheme for which no sub-exponential inverting algorithms are known involves taking discrete logarithms on an elliptic curve. Miller [M2] suggested the "supersingular" case, which is the simplest and most efficient; the interested reader should see [M1]. Everything is analogous to the previous section. *Generator*, on input (k,n), uniformly selects a k-bit prime p such that $p \equiv 3 \bmod 4$ and $d^{p+1,n} < n^3$. Let $Curve_p = \{(x,y) \in Z_p : y^2 \equiv x^3 + x \bmod p\}$ and a point called ∞. The points on $Curve_p$ form a cyclic group of order $p+1$. A generator $P \in Curve_p$ is chosen; $x = (p,P)$. For $y \in MES = Z_{p+1}$, let $Encrypt(y) = y \cdot P$. This satisfies the homomorphism property,

$Encrypt(B+C) = (B+C) \cdot P = Encrypt(B)\,Encrypt(C)$.

Set $e = p+1/d_{p+1,n}$. Let $Predicate(y)$ be the most significant $|k\ |$ bits of y mod e. By Kaliski [Ka], guessing *Hard* better than randomly is equivalent to computing discrete logs on (supersingular) elliptic curves. We could have *Generator* select a general elliptic curve (subject to some restrictions), but this would entail high (polynomial) computation and lengthier analysis.

8.3 Encryption Based on the Difficulty of Distinguishing r-th Powers

The following probabilistic encryption scheme was first suggested by Cohen and Fisher [CF] with application to a voting protocol; Benaloh [Be2] suggested its use for our VSS.

Generator, on input (n,k), uniformly selects an $|n\ |+1$-bit prime r, k-bit primes p and q such that $r\ |p-1$, and $z \leftarrow \{u \in Z_p^* : u^{(p-1)/r} \bmod p \neq 1\}$, that is, a non-$r$-th power modulo p. Set $m = pq$; $x = r,m,z$. Here, *Encrypt* itself is probabilistic. The domain of *Encrypt*, *MES*, is $[1,r]$; the range, *CYPH*, is Z_m^*. For $s \in [1,r]$, $Encrypt(s) \leftarrow \{z^s v^r \bmod m : v \in Z_m^*\}$; $Predicate(s) = s$. No way of computing *Hard* without knowing the factorization of m is known. To check the homomorphism property, let $(z^B v^r \bmod m) \leftarrow Encrypt(B)$ and $(z^C w^r \bmod m) \leftarrow Encrypt(C)$, and observe that their product, $(z^{B+C}(vw)^r \bmod m)$, is a possible encryption of $B+C$.

In this scheme, division by $n!$ is easy in *MES*. However, this method of probabilistic encryption requires the dealer to append to each piece M_l a string v_l enabling the recipient to verify equation 4.3, that the appropriate product of the encrypted coefficients is a possible encryption of M_l. A real problem is how to convince the players that z is not an r-th power without revealing the factorization of m. This apparently requires prefacing an interactive zero-knowledge protocol proving that z is not an r-th power.

8.4 RSA

As we mentioned in Section 4.1, the RSA encryption scheme behaves sufficiently homomorphic to be used in the VSS. An advantage of RSA is that numerous attempts to "break" the scheme have been made. The only successes have been for the special case of a very small exponent, which, we shall see, could not be used for VSS in any case. Alexi, Chor, Goldreich, and Schnorr [ACGS] prove a reduction from advantageously guessing *Hard* (as defined in Example 4.1) to inverting the RSA function *Encrypt*. They extend the reduction to the case where the predicate *Hard* contains $O(\ |k\ |)$ bits.

Generator, on input (k,n), uniformly selects k-bit primes p and q and a $2k$-bit prime e, $e > n$. Set $m = pq$; $x = e,m$. The domain and range of *Encrypt* are both Z_m^*. For $y \in Z_m^*$, $Encrypt(y) = y^e \bmod m$. The "addition" for RSA is multiplication mod m, so the homomorphism property is

$Encrypt(BC) = (BC)^e \bmod m = Encrypt(B)\,Encrypt(C)$.

Since $(e, \phi(m)) = 1$, *Encrypt* is 1-1. We set $Predicate(y)$ to be the $|k\ |$ least significant bits of y.

We defined $Divide(n, n! \cdot y) = Predicate(y)$. For RSA, this is inherently ill-defined, since $n! \cdot y = y^{n!} \bmod m = n! \cdot (m-y)$ but $Predicate(y) \neq Predicate(m-y)$ (exactly one of them is even). We remedy the situation by making $Encrypt(y)$ an additional input to *Divide*. Since $(e, n!) = 1$, we can find u,v such that $ue + v(n!) = 1$. Notice that $Encrypt(y)$ and $y^{n!} \bmod m$, uniquely determine y and show how to compute it easily: $y = y^{ue+vn!} = (Encrypt(y)^u)((y^{n!})^v) \bmod m$, so $Divide(n, n! \cdot y, Encrypt(y))$ is easily computable.

9. Other Variations

9.1 Basing the VSS on Multiple Homomorphic Encryption Functions

Note that one can easily combine the security of multiple homomorphic encryption schemes. For example to encode a secret s using *Encrypt* and *Encrypt'*, the dealer picks s' at random, shares the secret s' using *Encrypt'* and the secret $s \oplus s'$ using *Encrypt*. The good players can recover both s' and $s \oplus s'$, and hence s. The adversary cannot recover s unless he can invert both *Encrypt* and *Encrypt'*.

9.2 Efficiency

The communicational complexity of VSS for all homomorphic encryption schemes proposed is very reasonable. The dealer initiates *Share* by broadcasting his identity, the index of *Encrypt*, x, and an encryption of a secret, Y. In step 1, he broadcasts the encrypted A_j's. In round 3, he broadcasts encryptions of the other t coefficients, the Y_a's. Each broadcast has $O(nk)$ bits. Also, $O(k)$ bits are privately sent to each player. For all homomorphic encryption schemes proposed here, we feel that 1000 is a reasonable value for k given sufficient for discrete logs on elliptic curves.

We shall examine the computational complexity in the case where discrete logs in finite fields are used. We ignore the cost of finding p. This is a one-time cost which is not large if we allow non-uniform selection; it may be avoided entirely by choosing p from a list of known very large primes. In such a case the security of the protocol hinges on a bolder assumption than the discrete log assumption (equation 8.1).

The most costly part of the computations is exponentiation. Evaluating $Encrypt(x) = g^x \bmod p$ can be performed in at most $2|x| \leq 2|p| = 2k$ multiplications mod p and at most k additions. The complexity will be determined by the speed of multiplication, which we take to be $O(k \log k)$. The dealer must encrypt a total of $O(n)$ values, hence this can be done in $O(nk^2 \log k)$ steps. This term swallows the cost of randomly selecting the values and evaluating the polynomial at n points.

For the other players, the most costly steps are running *Checkpiece* and *Recover*. A reasonable way to compute the product $(Y_0) \cdot (Y_1^h) \cdots (Y_t^{h^t})$ is the following:

1. Set $x_0 = Y_t$.
2. Set $x_{l+1} = Y_{t-l-1} \cdot (x_l)^h$.
3. Reset l to $l+1$; if $l < t$ return to step 2.

The desired product is x_t. Since $|h| \leq \log n$, step 2 requires at most $O(\log n)$ multiplications, hence *Checkpiece* requires $O(n \log n + k)$ multiplications. (The k are for the exponentiation of the piece itself.) In *Recover*, finding $t+1$ good pieces can require running *Checkpiece* $2t$ times. Having found $t+1$ good pieces, a player can perform the polynomial interpolation using at most $O(n^2)$ multiplications and divisions. Thus, a player may recover the secret in $O(n(n \log n + k) k \log k)$ steps.

9.3 Must We Permute the Pieces?

We believe that our protocol may be simplified to 1 round of communication. The pieces were permuted to facilitate the zero-knowledge simulation in the proof; indeed, we do not know how to do this simulation in the simpler setting for a dynamic adversary. (The simulation for a static adversary is straightforward.) Nevertheless, we conjecture that the simplified protocol is secure against a dynamic adversary as well.

We see no way that knowledge of encryptions of the coefficients of Q, or any subset of the pieces, can make certain other pieces "more useful". In fact, we could try the simulation for the simplified protocol with respect to a dynamic adversary A, and we would succeed whenever we guessed the t players that A corrupts; however, this happens with probability inverse exponential in n. This would contradict a strong intractability assumption; that is, if we assumed that any algorithm inverting *Encrypt* with significant probability must run in time exponential in n. This may apply to discrete logs on elliptic curves whenever $k > n$, and to all the other schemes described when k is a sufficiently large function of n.

10. Conclusion

We have presented a noninteractive verifiable secret sharing, optimal in that it tolerates up to $(n-1)/2$ bad players, which is provably bitwise secure, assuming the intractability of taking discrete logarithms (or one of the other homomorphic encryption schemes). As we observed in Section 3.3, verifiable secret sharing can be used as a subroutine to simulate a simultaneous broadcast network, which makes protocols such as secret bidding as efficient as the best verifiable secret sharing. We have also remarked that our VSS enables a network to reach Byzantine Agreement in constant expected time without any preprocessing. Moreover, the work of [GMW] suggests that verifiable secret sharing may enter as a key subroutine in protecting any protocol from faulty players, making the search for an efficient one even more important. The bitwise communication and local computation complexity of our protocols are small enough, $O(nk)$ and $O((n \log n + k) nk \log k)$ respectively, making them feasible to implement. We believe that the proof technique introduced to prove security may have application to many other cryptographic protocols.

Acknowledgement: I would like to thank the many people who greatly contributed to the writeup of this paper. I participated in an interactive proof of the results of this paper with Silvio Micali, whose suggestions immeasurably helped both the technical proof and presentation. Josh Benaloh suggested basing the protocol on arbitrary encryption functions with the homomorphism property. Shafi Goldwasser helped simplify the

presentation by pointing out that certain complications were unnecessary. David Shmoys spent much time proofreading drafts so other readers wouldn't have to; the text was greatly clarified by his suggestions. Suggestions of Lance Fortnow and Joe Halpern have also been incorporated.

[ACGS] W. Alexi, B.Chor, O. Goldreich, and C. Schnorr, RSA/Rabin Bits are $1/2+1/poly(\log n)$ Secure, 1984 FOCS.

[Ba] E. Bach, How to Generate Random Integers With Known Factorization, 1983 STOC.

[Be] J. Benaloh, Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret, 1986 Crypto.

[Be2] J. Benaloh, personal communication.

[C] B. Coan, Achieving Consensus in Fault-Tolerant Distributed Computer Systems: Protocols, Lower Bounds, and Simulations, PhD. thesis, MIT, 1987.

[CC] B. Chor and B. Coan, A Simple and Efficient Randomized Byzantine Agreement Algorithm, IEEE Transactions on Software Engineering, Vol. SE-11, No.6 1985.

[CF] J. Cohen and M. Fischer, A Robust and Verifiable Cryptographically Secure Election Scheme, 1985 FOCS.

[CGMA] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults, 1985 FOCS.

[CR] B. Chor and M. Rabin, Achieving Independence in Logarithmic Number of Rounds, to appear in PODC 1987.

[F] P. Feldman, MIT thesis, to appear.

[FM] P. Feldman and S. Micali, Byzantine Agreement From Scratch in Constant Expected Time, manuscript.

[GM] S. Goldwasser and S. Micali, Probabilistic Encryption, JCSS, Vol. 28, No. 2, April 1984.

[GMW] O. Goldreich, S. Micali, and A. Wigderson, Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design, 1986 FOCS.

[Ka] B. Kaliski, MIT thesis, to appear.

[L] D. Long, The Security of Bits in the Discrete Logarithm, Princeton Thesis, 1984.

[LW] D. Long and A. Wigderson, How Discrete is the Discrete Log?, 1983 STOC.

[M1] V. Miller, Elliptic Curves and Cryptography, Crypto 1985.

[M2] V. Miller, personal communication.

[RSA] R. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signature and Public Key Cryptosystems, Comm. of the ACM, Vol. 21, Feb. 1978.

[S] A. Shamir, How to Share a Secret, CACM Vol.22 No.11, 1979.

[Y] A. Yao, Theory and Applications of Trapdoor Functions, 1982 FOCS.