CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 20
November 11, 2013

BBS Pseudorandom Sequence Generator

A cryptographically strong PRSG

We present a cryptographically strong pseudorandom sequence generator due to Blum, Blum, and Shub (BBS).

BBS is defined by a Blum integer $n = pq$ and an integer ℓ.

It maps strings in $\mathbf{Z}_n^*$ to strings in $\{0,1\}^\ell$.

Given a seed $s_0 \in \mathbf{Z}_n^*$, we define a sequence $s_1, s_2, s_3, \ldots, s_\ell$, where $s_i = s_{i-1}^2 \bmod n$ for $i = 1, \ldots, \ell$.

The ℓ-bit output sequence BBS(s_0) is $b_1, b_2, b_3, \ldots, b_\ell$, where $b_i = \mathrm{lsb}(s_i)$ is the least significant bit of s_i.

Security of BBS

In the next several slides, we show that BBS is secure.

The proof reduces the problem of predicting the output of BBS to the quadratic residue problem.

We finally show that if there is a judge that successfully distinguishes BBS(S) from U, then there is a feasible method for distinguishing quadratic residues from non-residues with Jacobi symbol 1.

Recall QR assumption and Blum integers

The security of BBS is based on the assumed difficulty of determining, for a given $a \in \mathbf{Z}_n^*$ with Jacobi symbol 1, whether or not a is a quadratic residue, i.e., whether or not $a \in \mathrm{QR}_n$.

Recall from Lecture 19 that a Blum prime is a prime $p \equiv 3 \pmod 4$, and a Blum integer is a number $n = pq$, where p and q are distinct Blum primes.

Also, Blum primes and Blum integers have the important property that every quadratic residue a has exactly one square root y which is itself a quadratic residue.

Call such a y the *principal square root* of a and denote it by $\sqrt{a} \pmod n$ or simply by $\sqrt{a}$ when it is clear that mod n is intended.

Blum integers and the Jacobi symbol

Fact

Let n be a Blum integer and $a \in \mathrm{QR}_n$. Then $\left(\frac{a}{n}\right) = \left(\frac{-a}{n}\right) = 1$.

Proof.

This follows from the fact that if a is a quadratic residue modulo a Blum prime, then $-a$ is a quadratic non-residue. Hence,

$$\left(\frac{a}{p}\right) = -\left(\frac{-a}{p}\right) \text{ and } \left(\frac{a}{q}\right) = -\left(\frac{-a}{q}\right), \text{ so}$$

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\cdot\left(\frac{a}{q}\right) = \left(-\left(\frac{-a}{p}\right)\right)\cdot\left(-\left(\frac{-a}{q}\right)\right) = \left(\frac{-a}{n}\right).$$

□

Blum integers and the least significant bit

The low-order bits of x mod n and $(-x)$ mod n always differ when n is odd.

Let $\mathrm{lsb}(x) = (x \bmod 2)$ be the least significant bit of integer x.

Fact

If n is odd, then $\mathrm{lsb}(x \bmod n) \oplus \mathrm{lsb}((-x) \bmod n) = 1$.

First-bit prediction

A *first-bit predictor with advantage* ϵ is a probabilistic polynomial time algorithm A that, given $b_2, \ldots, b_\ell$, correctly predicts b_1 with probability at least $1/2 + \epsilon$.

This is not sufficient to establish that the pseudorandom sequence BBS(S) is indistinguishable from the uniform random sequence U, but if it did not hold, there certainly would exist a distinguishing judge.

Namely, the judge that outputs 1 if $b_1 = A(b_2, \ldots, b_\ell)$ and 0 otherwise would output 1 with probability greater than $1/2 + \epsilon$ in the case that the sequence came from BBS(S) and would output 1 with probability exactly $1/2$ in the case that the sequence was truly random.

BBS has no first-bit predictor under the QR assumption

If BBS has a first-bit predictor A with advantage ϵ, then there is a probabilistic polynomial time algorithm Q for testing quadratic residuosity with the same accuracy.

Thus, if quadratic-residue-testing is "hard", then so is first-bit prediction for BBS.

Theorem

Let A be a first-bit predictor for BBS(S) with advantage ϵ. Then we can find an algorithm Q for testing whether a number x with Jacobi symbol 1 is a quadratic residue, and Q will be correct with probability at least $1/2 + \epsilon$.

Construction of Q

Assume that A predicts b_1 given $b_2, \ldots, b_\ell$.

Algorithm $Q(x)$ tests whether or not a number x with Jacobi symbol 1 is a quadratic residue modulo n.

It outputs 1 to mean $x \in \mathrm{QR}_n$ and 0 to mean $x \notin \mathrm{QR}_n$.

To $Q(x)$:

1. Let $\hat{s}_2 = x^2 \bmod n$.
2. Let $\hat{s}_i = \hat{s}_{i-1}^2 \bmod n$, for $i = 3, \ldots, \ell$.
3. Let $\hat{b}_1 = \mathrm{lsb}(x)$.
4. Let $\hat{b}_i = \mathrm{lsb}(\hat{s}_i)$, for $i = 2, \ldots, \ell$.
5. Let $c = A(\hat{b}_2, \ldots, \hat{b}_\ell)$.
6. If $c = \hat{b}_1$ then output 1; else output 0.

Why Q works

Since $\left(\frac{x}{n}\right) = 1$, then either x or $-x$ is a quadratic residue. Let s_0 be the principal square root of x or $-x$. Let $s_1, \ldots, s_\ell$ be the state sequence and $b_1, \ldots, b_\ell$ the corresponding output bits of BBS(s_0).

We have two cases.

Case 1: $x \in \mathrm{QR}_n$. Then $s_1 = x$, so the state sequence of BBS(s_0) is

$$s_1, s_2, \ldots, s_\ell = x, \hat{s}_2, \ldots, \hat{s}_\ell,$$

and the corresponding output sequence is

$$b_1, b_2, \ldots, b_\ell = \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_\ell.$$

Since $\hat{b}_1 = b_1$, $Q(x)$ correctly outputs 1 whenever A correctly predicts b_1. This happens with probability at least $1/2 + \epsilon$.

Why Q works (cont.)

Case 2: $x \in \mathrm{QNR}_n$, so $-x \in \mathrm{QR}_n$. Then $s_1 = -x$, so the state sequence of BBS(s_0) is

$$s_1, s_2, \ldots, s_\ell = -x, \hat{s}_2, \ldots, \hat{s}_\ell,$$

and the corresponding output sequence is

$$b_1, b_2, \ldots, b_\ell = \neg\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_\ell.$$

Since $\hat{b}_1 = \neg b_1$, $Q(x)$ correctly outputs 0 whenever A correctly predicts b_1. This happens with probability at least $1/2 + \epsilon$.
In both cases, $Q(x)$ gives the correct output with probability at least $1/2 + \epsilon$.

Bit-Prediction

Bit-prediction and statistical independence

One important property of the uniform distribution U on bit-strings $b_1, \ldots, b_\ell$ is that the individual bits are statistically independent from each other.

This means that the probability that a particular bit $b_i = 1$ is unaffected by the values of the other bits in the sequence.

Thus, any algorithm that attempts to predict b_i, even knowing other bits of the sequence, will be correct only $1/2$ of the time.

We now translate this property of unpredictability to pseudorandom sequences.

Next-bit prediction

First-bit prediction seems rather uninteresting because pseudorandom bits are usually generated in order.

However, we would like it to be difficult to predict the next bit given the bits that came before.

Algorithm A is an *ϵ-next-bit predictor for bit i* if

$$\Pr[\mathrm{A}(\mathrm{b}_1, \ldots, \mathrm{b}_{\mathrm{i}-1}) = \mathrm{b}_\mathrm{i}] \geq \frac{1}{2} + \epsilon$$

where $(b_1, \ldots, b_i) = G_i(S)$.

As before, S is uniformly distributed over $\mathcal{S}$, $G(S)$ is a random variable over the output strings of G, and $G_i(S)$ is the corresponding random variable on the length-i prefixes of $G(S)$.

Next-bit prediction and indistinguishability

Next-bit prediction is closely related to indistinguishability.

Roughly speaking, $G(S)$ has a next-bit predictor for some bit i iff $G(S)$ is distinguishable from U.

The precise definitions under which this theorem is true are subtle, for one must quantify both the amount of time the judge and next-bit predictor algorithms are permitted to run as well as how much better than chance the judgments or predictions must be in order to be considered a successful judge or next-bit predictor.

We defer the mathematics for now and focus instead on the intuitive concepts that underlie this theorem.

Building a judge from a next-bit predictor

Let A be an ϵ-next-bit predictor for G for some bit i.

Here's how to build a judge J that distinguishes $G(S)$ from U with advantage ϵ.

- J, given a sample x drawn from either $G(S)$ or from U, runs $A(x)$ to produce $\hat{b}_i$.
- If $\hat{b}_i = b_i$, then J outputs 1.
- Otherwise, J outputs 0.

The advantage of J

For samples from $G(S)$, the judge will output 1 with the same probability that A successfully predicts bit b_i, which is at least $1/2 + \epsilon$.

For sequences drawn from U, the judge will output 1 with probability exactly $1/2$.

Hence, the judge distinguishes $G(S)$ from U with advantage ϵ.

It follows that no cryptographically strong PRSG can have an ϵ-next-bit predictor.

In other words, no algorithm that attempts to predict the next bit can have more than a "small" advantage ϵ over chance.

Previous-bit prediction

Previous-bit prediction, while perhaps less natural, is analogous to next-bit prediction.

An *ϵ-previous-bit predictor for bit i* is a probabilistic polynomial time algorithm A that, given bits $b_{i+1}, \ldots, b_\ell$, correctly predicts b_i with probability at least $1/2 + \epsilon$.

As with next-bit predictors, if $G(S)$ has a previous-bit predictor for some bit b_j, then some judge distinguishes $G(S)$ from U.

Again, I am being vague with the exact conditions under which this is true.

Hence, $G(S)$ has an ϵ-next-bit predictor for some bit i if and only if it has an ϵ'-previous-bit predictor for some bit j (where ϵ and ϵ' are related but not necessarily equal).

Special case of $\ell = 2$

To give some intuition into why such a fact might be true, we look at the special case of $\ell = 2$, that is, of 2-bit sequences.

The probability distribution $G(S)$ can be described by four probabilities

$$p_{u,v} = \Pr[\mathrm{b}_1 = \mathrm{u} \wedge \mathrm{b}_2 = \mathrm{v}], \text{ where } \mathrm{u}, \mathrm{v} \in \{0, 1\}.$$

Written in tabular form, we have

		b_2 = 0	b_2 = 1
b_1	0	$p_{0,0}$	$p_{0,1}$
	1	$p_{1,0}$	$p_{1,1}$

Bit prediction when $\ell = 2$

We describe a deterministic algorithm $A(v)$ for predicting b_1 given $b_2 = v$. $A(v)$ predicts $b_1 = 0$ if $p_{0,v} > p_{1,v}$, and it predicts $b_1 = 1$ if $p_{0,v} \leq p_{1,v}$.

In other words, the algorithm chooses the value for b_1 that is most likely given that $b_2 = v$.

Theorem

If A is an ϵ-previous-bit predictor for b_1, then A is an ϵ-next-bit predictor for either b_1 or b_2.

Proof that A is a next-bit predictor

Assume A is an ϵ-previous-bit predictor for b_1, so A correctly predicts b_1 given b_2 with probability $\geq 1/2 + \epsilon$.

We show that A is an ϵ-next-bit predictor for either b_1 or b_2.

Let $a(v)$ be the value predicted by $A(v)$ for $v \in \{0, 1\}$.

We have two cases:

Case 1: $a(0) = a(1)$. Then algorithm A does not depend on v, that is, it makes the same prediction regardless of the value of v.

Thus, $A(0)$ correctly predicts b_1 with probability at least $1/2 + \epsilon$. (This means that $\Pr[\mathrm{b}_1 = \mathrm{A}(0)] \geq 1/2 + \epsilon$.)

It follows that A is an ϵ-next-bit predictor for b_1.

Proof that A is a next-bit predictor (cont.)

Case 2: $a(0) \neq a(1)$. The probability that $A(v)$ correctly predicts b_1 given $b_2 = v$ is

$$\Pr[\mathrm{b}_1 = \mathrm{a}(\mathrm{v}) \mid \mathrm{b}_2 = \mathrm{v}] = \frac{\Pr[\mathrm{b}_1 = \mathrm{a}(\mathrm{v}) \wedge \mathrm{b}_2 = \mathrm{v}]}{\Pr[\mathrm{b}_2 = \mathrm{v}]} = \frac{\mathrm{p}_{\mathrm{a}(\mathrm{v}),\mathrm{v}}}{\Pr[\mathrm{b}_2 = \mathrm{v}]}$$

The overall probability that $A(b_2)$ is correct for b_1 is the average of the conditional probabilities for $v = 0$ and $v = 1$, weighted by the probability that $b_2 = v$. Thus,

$$\begin{aligned}
&\Pr[A(b_2) \text{ is correct for } b_1] \\
&= \sum_{v \in \{0,1\}} \Pr[\mathrm{b}_1 = \mathrm{a}(\mathrm{v}) \mid \mathrm{b}_2 = \mathrm{v}] \cdot \Pr[\mathrm{b}_2 = \mathrm{v}] \\
&= \sum_{v \in \{0,1\}} p_{a(v),v} = p_{a(0),0} + p_{a(1),1}
\end{aligned}$$

Proof that A is a next-bit predictor (cont.)

Similarly, using A to predict b_2 given b_1 yields

$$\begin{aligned}
&\Pr[A(b_1) \text{ is correct for } b_2] \\
&= \sum_{v \in \{0,1\}} \Pr[\mathrm{b}_2 = \mathrm{a}(\mathrm{u}) \mid \mathrm{b}_1 = \mathrm{u}] \cdot \Pr[\mathrm{b}_1 = \mathrm{u}] \\
&= \sum_{v \in \{0,1\}} p_{u,a(u)} = p_{0,a(0)} + p_{1,a(1)}
\end{aligned}$$

We show that

$$p_{a(0),0} + p_{a(1),1} = p_{0,a(0)} + p_{1,a(1)}$$

when $a(0) \neq a(1)$. It follows that

$$\Pr[A(b_1) \text{ is correct for } b_2] = \Pr[A(b_2) \text{ is correct for } b_1],$$

so A is an ϵ-next-bit predictor for b_2.

Proof that A is a next-bit predictor (cont.)

Since $a(0) \neq a(1)$, the function $a(\cdot)$ is one-to-one and onto, so either $a(v) = v$ for $v \in \{0, 1\}$, or $a(v) = \neg v$ for $v \in \{0, 1\}$.

That is, $a(\cdot)$ is either the identity or the complement function. Hence, either

$$p_{a(0),0} + p_{a(1),1} = p_{0,0} + p_{1,1} = p_{0,a(0)} + p_{1,a(1)}$$

or

$$p_{a(0),0} + p_{a(1),1} = p_{1,0} + p_{0,1} = p_{0,a(0)} + p_{1,a(1)}$$

as desired. Hence, A is an ϵ-next-bit predictor for b_2.

Combining the two cases, we conclude that A is an ϵ-next-bit predictor for either b_1 or b_2, proving the theorem.

Summary of results

We have just seen how to construct a next-bit predictor from a previous-bit predictor, and we've also seen how to construct a judge from a next-bit predictor.

The most general bit-prediction problem is to predict the i^{th} bit of the sequence given *all* other bits. An algorithm that can do this with advantage ϵ is said to be an *ϵ-i^{th}-bit predictor for G*.

It's easy to transform an ϵ-next-bit predictor for b_i into an ϵ-i^{th}-bit predictor.

It's also easy to build a judge with advantage ϵ from an ϵ-i^{th}-bit predictor.

To close the loop, one can build an ϵ'-next-bit predictor for some bit i and some ϵ' given a judge with advantage ϵ.

Bit-prediction given a judge

We sketch how to build a next-bit predictor given a judge.
The construction is based on interpolation between U and $G(S)$.

$$
\begin{array}{cccccccccc}
u_1 & u_2 & u_3 & \dots & u_{i-1} & u_i & u_{i+1} & \dots & u_\ell \\
b_1 & u_2 & u_3 & \dots & u_{i-1} & u_i & u_{i+1} & \dots & u_\ell \\
 & & & \dots & & & & \dots & \\
b_1 & b_2 & b_3 & \dots & b_{i-1} & u_i & u_{i+1} & \dots & u_\ell \\
b_1 & b_2 & b_3 & \dots & b_{i-1} & b_i & u_{i+1} & \dots & u_\ell \\
 & & & \dots & & & & \dots & \\
b_1 & b_2 & b_3 & \dots & b_{i-1} & b_i & b_{i+1} & \dots & b_\ell
\end{array}
$$

The difference in the judge's output between top and bottom sequence is $\geq \epsilon$.

Therefore, for some i, the difference in judge's output between sequence $i-1$ and i must be at least $\epsilon' = \epsilon/\ell$.

An ϵ'-next bit predictor for b_i is easily constructed.

Secret Splitting

Two-key locks

There are many situations in which one wants to grant access to a resource only if a sufficiently large group of agents cooperate.

For example, the office safe of a supermarket might require both the manager's key and the armored car driver's key in order to be opened.

This protects the store against a dishonest manager or armored car driver, and it also prevents an armed robber from coercing the manager into opening the safe.

A similar 2-key system is used for safe deposit boxes in banks.

Two-part secret splitting

We might like to achieve the same properties for cryptographic keys or other secrets. (This concept was introduced in Lecture 17.)

Let k be the key for a symmetric cryptosystem. One might wish to split k into two *shares* k_1 and k_2 so that by themselves, neither k_1 nor k_2 by itself reveals any information about k, but when suitably combined, k can be recovered.

A simple way to do this is to choose k_1 uniformly at random and then let $k_2 = k \oplus k_1$.

Both k_1 and k_2 are uniformly distributed over the key space and hence give no information about k.

However, combined with XOR, they reveal k, since $k = k_1 \oplus k_2$.

Comparison with one-time pad

Indeed, the one-time pad cryptosystem of Lecture 3 can be viewed as an instance of secret splitting.

Here, Alice's secret is her message m.

The two shares are the ciphertext c and the key k.

Neither by themselves gives any information about m, but together they reveal $m = k \oplus c$.

Multi-part secret splitting

Secret splitting generalizes to more than two shares.

Imagine a large company that restricts access to important company secrets to only its five top executives, say the president, vice-president, treasurer, CEO, and CIO.

They don't want any executive to be able to access the data alone since they are concerned that an executive might be blackmailed into giving confidential data to a competitor.

Multi-part secret splitting (cont.)

On the other hand, they also don't want to require that all five executives get together to access their data because

- this would be cumbersome;
- they worry about the death or incapacitation of any single individual.

They decide as a compromise that any three of them should be able to access the secret data, but one or two of them operating alone should not have access.

Shamir's Secret Splitting Scheme

(τ, k) threshold secret spitting scheme

A (τ, k) *threshold secret splitting scheme* splits a secret s into *shares* $s_1, \ldots, s_k$.

Any subset of τ or more shares allows s to be recovered, but no subset of shares of size less than τ gives any information about s.

The executives of the previous example thus want a $(3, 5)$ threshold secret splitting scheme: The secret key is to be split into 5 shares, any 3 of which allow the secret to be recovered.

A threshold scheme based on polynomials

Shamir proposed a threshold scheme based on polynomials.

A *polynomial of degree d* is an expression

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d,$$

where $a_d \neq 0$.

The numbers $a_0, \ldots, a_d$ are called the *coefficients* of f.

A polynomial can be simultaneously regarded as a function and as an object determined by its vector of coefficients.

Interpolation

Interpolation is the process of finding a polynomial that goes through a given set of points.

Fact

Let $(x_1, y_1), \ldots, (x_k, y_k)$ be points, where all of the x_i's are distinct. There is a unique polynomial $f(x)$ of degree at most $k-1$ that passes through all k points, that is, for which $f(x_i) = y_i$ $(1 \leq 1 \leq k)$.

f can be found using Lagrangian interpolation. This statement generalizes the familiar statement from high school geometry that two points determine a line.

Lagrangian interpolation method

One way to understand Lagrangian interpolation is to consider the polynomial

$$\delta_i(x) = \frac{(x - x_1)(x - x_2)\dots(x - x_{i-1}) \cdot (x - x_{i+1})\dots(x - x_k)}{(x_i - x_1)(x_i - x_2)\dots(x_i - x_{i-1}) \cdot (x_i - x_{i+1})\dots(x_i - x_k)}$$

Although this looks at first like a rational function, it is actually just a polynomial in x since the denominator contains only the x-values of the given points and not the variable x.

$\delta_i(x)$ has the easily-checked property that $\delta_i(x_i) = 1$, and $\delta_i(x_j) = 0$ for $j \neq i$.

Lagrangian interpolation method (cont.)

Using $\delta_i(x)$, the polynomial

$$p(x) = \sum_{i=1}^{k} y_i \, \delta_i(x)$$

is the desired interpolating polynomial, since $p(x_i) = y_i$ for $i = 1, \ldots, k$.

To actually find the coefficients of $p(x)$ when written as

$$p(x) = \sum_{i=0}^{k} a_i x^i,$$

it is necessary to expand $p(x)$ by multiplying out the factors and collect like terms.

Interpolation over finite fields

Interpolation also works over finite fields such as $\mathbf{Z}_p$ for prime p.

It is still true that any k points with distinct x coordinates determine a unique polynomial of degree at most $k-1$ over $\mathbf{Z}_p$.

Of course, we must have $k \leq p$ since $\mathbf{Z}_p$ has only p distinct coordinate values in all.

Shamir's secret splitting scheme

Here's how Shamir's (τ, k) secret splitting scheme works.

Let Alice (also called the *dealer*) have secret s.

She first chooses a prime $p > k$ and announces it to all players.

Constructing the polynomial

She next constructs a polynomial

$$f = a_0 + a_1 x + a_2 x^2 \ldots a_{\tau-1} x^{\tau-1}$$

of degree at most $\tau - 1$ as follows:

- She sets $a_0 = s$ (the secret).
- She chooses $a_1, \ldots, a_{\tau-1} \in Z_p$ at random.

Constructing the shares

She constructs the k shares as follows:

- She chooses $x_i = i$. $\quad (1 \leq i \leq k)$
- She chooses $y_i = f(i)$. $\quad (1 \leq i \leq k)$[1]
- Share $s_i = (x_i, y_i) = (i, f(i))$.

[1] $f(i)$ is the result of evaluating the polynomial f at the value $x = i$. All arithmetic is over the field $\mathbf{Z}_p$, so we omit explicit mention of mod p.

Recovering the secret

Theorem

s can be reconstructed from any set T of τ or more shares.

Proof.

Suppose $s_{i_1}, \ldots, s_{i_\tau}$ are τ distinct shares in T.

By interpolation, there is a unique polynomial $g(x)$ of degree $d \leq \tau - 1$ that passes through these shares.

By construction of the shares, $f(x)$ also passes through these same shares; hence $g = f$ as polynomials.

In particular, $g(0) = f(0) = s$ is the secret. □

Protection from unauthorized disclosure

Theorem

For any set T' of fewer than τ shares and any possible secret $\hat{s}$, there is a polynomial $\hat{f}$ that interprets those shares and reveals $\hat{s}$.

Proof.

Let $T' = \{s_{i_1}, \ldots, s_{i_r}\}$ be a set of $r < \tau$ shares.

In particular, for each $s' \in \mathbf{Z}_p$, there is a polynomial $g_{s'}$ that interpolates the shares in $T' \cup \{(0, s')\}$.

Each of these polynomials passes through all of the shares in T', so each is a plausible candidate for f. Moreover, $g_{s'}(0) = s'$, so each s' is a plausible candidate for the secret s. □

No information about secret

One can show further that the number of polynomials that interpolate $T' \cup \{(0, s')\}$ is the same for each $s' \in \mathbf{Z}_p$, so each possible candidate s' is equally likely to be s.

Hence, the shares in T' give no information at all about s.

Secret splitting with semi-honest parties

Shamir's scheme is an example of a protocol that works assuming *semi-honest* parties.

These are players that follow the protocol but additionally may collude in an attempt to discover secret information.

We just saw that no coalition of fewer than τ players could learn anything about the dealer's secret, even if they pooled all of their shares.

Secret splitting with dishonest dealer

In practice, either the dealer or some of the players (or both) may be dishonest and fail to follow the protocol. The honest players would like some guarantees even in such situations.

A dishonest dealer can always lie about the true value of her secret. Even so, the honest players want assurance that their shares do in fact encode a unique secret, that is, all sets of τ shares reconstruct the same secret s.

Failure of Shamir's scheme with dishonest dealer

Shamir's (τ, k) threshold scheme assumes that all k shares lie on a single polynomial of degree at most $\tau - 1$.

This might not hold if the dealer is dishonest and gives bad shares to some of the players.

The players have no way to discover that they have bad shares until later when they try to reconstruct s, and maybe not even then.

Verifiable secret sharing

In *verifiable secret sharing*, the sharing phase is an active protocol involving the dealer and all of the players.

At the end of this phase, either the dealer is exposed as being dishonest, or all of the players end up with shares that are consistent with a single secret.

Needless to say, protocols for verifiable secret sharing are quite complicated.

Dishonest players

Dishonest players present another kind of problem. These are players that fail to follow the protocol. During the reconstruction phase, they may fail to supply their share, or they may present a (possibly different) corrupted share to each other player.

With Shamir's scheme, a share that just disappears does not prevent the secret from being reconstructed, as long as enough valid shares remain.

But a player who lies about his share during the reconstruction phase can cause other players to reconstruct incorrect values for the secret.

Fault-tolerance in secret sharing protocols

A *fault-tolerant secret sharing scheme* should allow the secret to be correctly reconstructed, even in the face of a certain number of corrupted shares.

Of course, it may be desirable to have schemes that can tolerate dishonesty in both dealer and a limited number of players.

The interested reader is encouraged to explore the extensive literature on this subject.