CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Preview Lecture 21
November 15, 2017
Secure random sequence generators
   Pseudorandom sequence generators
   Looking random

Similarity of Probability Distributions
   Cryptographically secure PRSG
   Indistinguishability

Secret Splitting

Shamir’s Secret Splitting Scheme
   Secret Splitting with Dishonest Parties
Secure random sequence generators
A *pseudorandom sequence generator (PRSG)* is a function that maps a short *seed* to a long “random-looking” *output sequence*.

The seed typically has length between 32 and a few thousand bits.

The output is typically much longer, ranging from thousands or millions of bits or more, but polynomially related to the seed length.

The output of a PRSG is a sequence that is supposed to “look random”.
Incremental generators

In practice, a PRSG is implemented as a co-routine that outputs the next block of bits in the sequence each time it is called. For example, the Linux function

```c
void srandom(unsigned int seed)
```

sets the 32-bit seed. Each subsequent call on

```c
long int random(void)
```

returns an integer in the range \([0, \ldots, \text{RAND\_MAX}]\).

On my machine, the return value is 31 bits long (even though `sizeof(long int)` is 64).
Limits on incremental generators

Incremental generators typically are based on state machines with a finite number of states, so the output eventually becomes periodic.

The period of `random()` is said to be approximately $16 \times (2^{31} - 1)$.

The output of a PRSG becomes predictable from past outputs once the generator starts to repeat. The point of repetition defines the *maximum usable output length*, even if the implementation allows bits to continue to be produced.
What does it mean for a string to look random?

For the output of a PRSG to look random:

- It must pass common statistical tests of randomness. For example, the frequencies of 0’s and 1’s in the output sequence must be approximately equal.
- It must lack obvious structure, such as having all 1’s occur in pairs.
- It must be difficult to find the seed given the output sequence, since otherwise the whole sequence is easily generated.
- It must be difficult to correctly predict any generated bit, even knowing all of the other bits of the output sequence.
- It must be difficult to distinguish its output from truly random bits.
Chaitin/Kolmogorov randomness

Chaitin and Kolmogorov defined a string to be “random” if its shortest description is almost as long as the string itself.

By this definition, most strings are random by a simple counting argument.

For example, \(011011011011011011011011011\) is easily described as the pattern \(011\) repeated 9 times. On the other hand, \(101110100010100101001000001\) has no obvious short description.

While philosophically very interesting, these notions are somewhat different than the statistical notions that most people mean by randomness and do not seem to be useful for cryptography.
Cryptographically secure PRSG

A PRSG is said to be *cryptographically secure* if its output cannot be *feasibly* distinguished from truly random bits.

In other words, no feasible probabilistic algorithm behaves significantly differently when presented with an output from the PRSG as it does when presented with a truly random string of the same length.

We argue that this definition encompasses all of the desired properties for “looking random” discussed earlier,
Looking ahead

In the rest of this lecture, we carefully define what it means for a PRSG to be secure.

We then show how to build a PRSG that is provably secure. It is based on the *quadratic residuosity assumption* (lecture 20) on which the Goldwasser-Micali probabilistic cryptosystem is based.
Similarity of Probability Distributions
Formal definition of PRSG

Formally, a pseudorandom sequence generator $G$ is a function from a domain of seeds $S$ to a domain of strings $\mathcal{X}$.

We generally assume that all of the seeds in $S$ have the same length $n$ and that $\mathcal{X}$ is the set of all binary strings of length $\ell = \ell(n)$.

$\ell(\cdot)$ is called the expansion factor of $G$.

$\ell(\cdot)$ is assumed to be a polynomial such that $n \ll \ell(n)$.
Output distribution of a PRSG

Let $S$ be a uniformly distributed random variable over the set $S$ of possible seeds.

The **output distribution** of $G$ is a random variable $X \in \mathcal{X}$ defined by $X = G(S)$.

For $x \in \mathcal{X}$,

$$\Pr[X = x] = \frac{|\{s \in S \mid G(s) = x\}|}{|S|}.$$ 

Thus, $\Pr[X = x]$ is the probability of obtaining $x$ as the output of the PRSG for a randomly chosen seed.
Cryptographically secure PRSG

Randomness amplifier

We think of $G(\cdot)$ as a *randomness amplifier*.

We start with a short truly random seed and obtain a long random string distributed according to $X$, which is very much non-uniform.

Because $|S| \leq 2^n$, $|X| = 2^\ell$, and $n \ll \ell$, most strings in $X$ are not in the range of $G$ and hence have probability 0.

For the uniform distribution $U$ over $X$, all strings have the same non-zero probability $1/2^\ell$.

$U$ is what we usually mean by a *truly random* variable on $\ell$-bit strings.
Indistinguishability

Computational indistinguishability

We have just seen that the probability distributions of $X = G(S)$ and $U$ are quite different.

Nevertheless, it may be the case that all feasible probabilistic algorithms behave essentially the same whether given a sample chosen according to $X$ or a sample chosen according to $U$.

If that is the case, we say that $X$ and $U$ are *computationally indistinguishable* and that $G$ is a *cryptographically secure* pseudorandom sequence generator.
Some implications of computational indistinguishability

Before going further, let me describe some functions $G$ for which $G(S)$ is readily distinguished from $U$.

Suppose every string $x = G(s)$ has the form $b_1 b_1 b_2 b_2 b_3 b_3 \ldots$, for example $00111110001100110000 \ldots$.

Algorithm $A(x)$ outputs “G” if $x$ is of the special form above, and it outputs “U” otherwise.

$A$ will always output “G” for inputs from $G(S)$. For inputs from $U$, $A$ will output “G” with probability only

$$\frac{2^{\ell/2}}{2^\ell} = \frac{1}{2^{\ell/2}}.$$

How many strings of length $\ell$ have the special form above?
Indistinguishability

Judges

Formally, a judge is a probabilistic polynomial-time algorithm $J$ that takes an $\ell$-bit input string $x$ and outputs a single bit $b$.

Thus, it defines a probabilistic function from $\mathcal{X}$ to $\{0, 1\}$.

This means that for every input $x$, the output is 1 with some probability $p_x$, and the output is 0 with probability $1 - p_x$.

If the input string is a random variable $X$, then the probability that the output is 1 is the weighted sum of $p_x$ over all possible inputs $x$, where the weight is the probability $\Pr[X = x]$ of input $x$ occurring.

Thus, the output value is itself a random variable $J(X)$, where

$$\Pr[J(X) = 1] = \sum_{x \in \mathcal{X}} \Pr[X = x] \cdot p_x.$$
Formal definition of indistinguishability

Two random variables $X$ and $Y$ are $\epsilon$-indistinguishable by judge $J$ if

$$\left| \Pr[J(X) = 1] - \Pr[J(Y) = 1] \right| < \epsilon.$$ 

Intuitively, we say that $G$ is cryptographically secure if $G(S)$ and $U$ are $\epsilon$-indistinguishable for suitably small $\epsilon$ by all judges that do not run for too long.

A careful mathematical treatment of the concept of indistinguishability must relate the length parameters $n$ and $\ell$, the error parameter $\epsilon$, and the allowed running time of the judges.

Further formal details may be found in Goldwasser and Bellare and in handout 12.
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 19.)

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 in the appendix 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-share 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-share 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
A \((\tau, k)\) threshold secret splitting scheme splits a secret \(s\) into shares \(s_1, \ldots, s_k\).

Any subset of \(\tau\) or more shares allows \(s\) to be recovered, but no subset of shares of size less than \(\tau\) 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

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

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 $\mathbb{Z}_p$ 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 $\mathbb{Z}_p$. Of course, we must have $k \leq p$ since $\mathbb{Z}_p$ has only $p$ distinct coordinate values in all.
Shamir’s secret splitting scheme

Here’s how Shamir’s \((\tau, k)\) secret splitting scheme works.

Let Alice (also called the \textit{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

\[ f = a_0 + a_1x + a_2x^2 \ldots a_{\tau - 1}x^{\tau - 1} \]

of degree at most \( \tau - 1 \) as follows:

- She sets \( a_0 = s \) (the secret).
- She chooses \( a_1, \ldots, a_{\tau - 1} \in \mathbb{Z}_p \) 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 $y_i = f(i). \quad (1 \leq i \leq k)$
- Share $s_i = (x_i, y_i) = (i, f(i))$.

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

**Theorem**

$s$ can be reconstructed from any set $T$ of $\tau$ or more shares.

**Proof.**

Suppose $s_{i_1}, \ldots, s_{i_\tau}$ are $\tau$ 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.

\[ \square \]
Protection from unauthorized disclosure

Theorem
For any set $T'$ of fewer than $\tau$ 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 $s' \in \mathbb{Z}_p$, 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 $s$. $\square$
No information about secret

One can show further that the number of polynomials that interpolate $T' \cup \{(0, s')\}$ is the same for each $s' \in \mathbb{Z}_p$, 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 $\tau$ 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 $\tau$ shares reconstruct the same secret $s$. 
Failure of Shamir’s scheme with dishonest dealer

Shamir’s \((\tau, k)\) 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.