CPSC 467: Cryptography and Security

Michael J. Fischer

Lecture 21
November 12, 2020

BBS Pseudorandom Sequence Generator

A cryptographically secure PRSG

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

Its security is based on the quadratic residuosity assumption.

The number theory that it is based on and the proof that it is cryptographically secure are given in the supplement to this lecture.

Blum primes and integers

BBS uses a special kind of prime called a *Blum prime*. This is a prime number p such that $p \equiv 3 \pmod 4$.

A *Blum integer* is a number $n = pq$, where p and q are distinct Blum primes.

If p is a Blum prime, then $-1 \in \mathrm{QNR}_p$. This follows from the Euler criterion, since $\frac{p-1}{2}$ is odd.

If n is a Blum integer, then $-1 \in \mathrm{QNR}_n$.

Remarkable property: If n is a Blum integer, then among the four square roots of $a \pmod n$, *exactly one* is itself a quadratic residue modulo n, so the squaring function has a unique inverse in QR_n.

The Blum-Blum-Shub PRSG

Here's the deceptively simple algorithm!

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

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

Given a seed $s_0 \in \mathsf{Z}_n^*$, 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 $\mathrm{BBS}(s_0)$ 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.

Remarks about BBS

- The overall structure is the same as a linear congruential random number generator except that the function for generating the next state is not linear but quadratic, and only one bit is extracted from each state.
- This is similar to an old idea based on squaring but without the use of a Blum integer as a modulus.
- Every output bit is unpredictable, even given all of the other output bits.
- Special case: After Mallory has seen $b_1, \ldots, b_k$, she still has at most a negligible advantage at guessing the next bit b_{k+1}.

Protocols Among Distrustful Parties

Privacy

We have looked at several protocols that are intended to keep Alice's information secret, both from Bob and from a malicious adversary masquerading as Bob.

We now look at other protocols whose goal is to control the release of partial information about Alice's secret. Just enough information should be released to carry out the purpose of the protocol but no more.

We'll see several examples in the following sections.

Commitment Schemes

Bit guessing game

Alice and Bob want to play a guessing game over the internet.

Alice says,

> *"I'm thinking of a bit. If you guess my bit correctly, I'll give you $10. If you guess wrong, you give me $10."*

Bob says,

> *"Ok, I guess zero."*

Alice replies,

> *"Sorry, you lose. I was thinking of one."*

Preventing Alice from changing her mind

While this game may seem fair on the surface, there is nothing to prevent Alice from changing her mind after Bob makes his guess.

Even if Alice and Bob play the game face to face, they still must do something to *commit* Alice to her bit before Bob makes his guess.

For example, Alice might be required to write her bit down on a piece of paper and seal it in an envelope.

After Bob makes his guess, he opens the envelope to know whether he won or lost.

Writing down the bit commits Alice to that bit, even though Bob doesn't learn its value until later.

Cryptographic envelopes

A *commitment* or *blob* or *cryptographic envelope* is an electronic analog of a sealed envelope.

The secret inside the blob has two properties:

1. It remains hidden until the blob is opened.
2. It cannot be changed once the envelope is sealed, that is, the blob cannot be opened in different ways to reveal different valid secrets.

Commitment schemes

A *commitment* c to a secret s is an "encryption" of s using a cryptosystem with two special properties:

1. [*Hiding*] The secret s cannot be found from c by anyone not knowing the secret key.
2. [*Binding*] All keys k' that decrypt c to a valid secret reveal the same secret.

Thus, if $c = E_k(s)$:

- It is hard to find s from c without knowing k.
- For every k', s', if $E_{k'}(s') = c$, then $s' = s$.

Commitment intuition

In other words,

- If Alice produces a commitment c to a secret s, then s cannot be recovered from c without knowing Alice's secret key k.
- There is no key k' that Alice might release that would make it appear that c is a commitment of some other secret $s' \neq s$, so Alice cannot pretend her secret was something other than s.

Commitment primitives

A blob is produced by a protocol commit(s) between Alice and Bob. We assume initially that only Alice knows s.

At the end of the commit protocol, Bob has a blob c containing Alice's secret s, but he should have no information about s's value.

Later, Alice and Bob can run a protocol open(c) to reveal the secret contained in c to Bob.

Requirements for secret commitment

Alice and Bob do not trust each other, so each wants protection from cheating by the other.

- Alice wants to be sure that Bob cannot learn s after she runs commit(s), even if he cheats.
- Bob wants to be sure that all successful runs of open(c) reveal the same secret s', no matter what Alice does.

We do *not* require that Alice tell the truth about her private secret s. A dishonest Alice can always pretend her secret was $s' \neq s$ when producing c. But if she does, c can only be opened to s', not to s.

These ideas should become clearer in the protocols below.

Bit Commitment Using QR Cryptosystem

A simple (but inefficient) bit commitment scheme

Here's a simple way for Alice to commit to a secret that is a single bit b.

1. Create a Goldwasser-Micali public key $e = (n, y)$, where $n = pq$.
2. Choose random $r \in \mathsf{Z}_n^*$, and use it to produce an encryption c of b. (See lecture 20b.)
3. Send the blob (e, c) to Bob.

To open (e, c), Alice sends b, p, q, r to Bob.

Bob checks that $n = pq$, p and q are distinct odd primes, $y \in Q_n^{00}$, and that c is the Goldwasser-Micali encryption of b that results from random choice r.

Security of QR bit commitment

Alice can't change her mind about b, since c either is or is not a quadratic residue.

Bob cannot determine the value of b before Alice opens c since that would amount to violating the quadratic residuosity assumption.

Bit Commitment Using Hash Functions

Bit commitment from a hash function

The analogy between bit commitment and hash functions described above suggests a bit commitment scheme based on hash functions.

	Alice		Bob
	To commit(b):		
1.		$\xleftarrow{r_1}$	Choose random string r_1.
2.	Choose random string r_2.		
	Compute $c = H(r_1 r_2 b)$.	$\xrightarrow{c}$	c is commitment.
	To open(c):		
3.	Send r_2.	$\xrightarrow{r_2}$	Find $b' \in \{0,1\}$ such that $c = H(r_1 r_2 b')$. If no such b', then fail. Otherwise, b' is revealed bit.

Purpose of r_2

The purpose of r_2 is to protect Alice's secret bit b.

To find b before Alice opens the commitment, Bob would have to find r_2' and b' such that $H(r_1 r_2' b') = c$.

This is akin to the problem of inverting H and is likely to be hard, although the one-way property for H is not strong enough to imply this.

On the one hand, if Bob succeeds in finding such r_2' and b', he has indeed inverted H, but he does so only with the help of r_1 — information that is not generally available when attempting to invert H.

Purpose of r_1

The purpose of r_1 is to strengthen the protection that Bob gets from the hash properties of H.

Even without r_1, the strong collision-free property of H would imply that Alice cannot find c, r_2, and r_2' such that $H(r_2 0) = c = H(r_2' 1)$.

But by using r_1, Alice would have to find a new colliding pair for each run of the protocol.

This protects Bob by preventing Alice from exploiting a few colliding pairs for H that she might happen to discover.

Bit Commitment Using Pseudorandom Sequence Generators

Cryptographically strong PRSG

Recall that a PseudoRandom Sequence Generator (PRSG) is a function G that maps short seeds to long pseudorandom output sequences.

G is *cryptographically strong* if the behavior of every probabilistic polynomial time algorithm J (the "judge") is essentially the same, whether J's source of random bits is a sequence of coin flips from a fair coin or from the bits of the sequence $G(s)$ that results from a truly random seed s.

We can build a bit-commitment scheme using such a G.

Bit commitment using a PRSG

Let $G_\rho(s)$ be the first ρ bits of $G(s)$. (ρ is a security parameter.)

	Alice		Bob
	To commit(b):		
1.		$\xleftarrow{r}$	Choose random $r \in \{0,1\}^\rho$.
2.	Choose random seed s.		
	Let $y = G_\rho(s)$.		
	If $b = 0$ let $c = y$.		
	If $b = 1$ let $c = y \oplus r$.	$\xrightarrow{c}$	c is commitment.
	To open(c):		
3.	Send s.	$\xrightarrow{s}$	Let $y = G_\rho(s)$.
			If $c = y$ then reveal 0.
			If $c = y \oplus r$ then reveal 1.
			Otherwise, fail.

Why this protocol works

We argue informally why this is a proper bit-commitment protocol.

- If Bob is able to get an advantage at predicting b knowing only c and r, then he has an advantage at predicting the first ρ bits of $G(s)$. This contradicts the assumption that G is cryptographically strong.
- If Alice can open c to reveal either 0 or 1, then she has found two seeds s_0 and s_1 such that $G_\rho(s_0) \oplus G_\rho(s_1) = r$. This is not possible for most values of r if ρ is sufficiently large.

Details are in the appendix.

Coin-Flipping

Flipping a common coin

Alice and Bob are in the process of getting a divorce and are trying to decide who gets custody of their pet cat, Fluffy.

They both want the cat, so they agree to decide by flipping a coin: heads Alice wins; tails Bob wins.

Bob has already moved out and does not wish to be in the same room with Alice.

The feeling is mutual, so Alice proposes that she flip the coin and telephone Bob with the result.

This proposal of course is not acceptable to Bob since he has no way of knowing whether Alice is telling the truth when she says that the coin landed heads.

Making it fair

"Look Alice," he says, "to be fair, we both have to be involved in flipping the coin."

"We'll each flip a private coin and XOR our two coins together to determine who gets Fluffy."

"You should be happy with this arrangement since even if you don't trust me to flip fairly, your own fair coin is sufficient to ensure that the XOR is unbiased."

A proposed protocol

This sounds reasonable to Alice, so she lets him propose the protocol below, where 1 means "heads" and 0 means "tails".

	Alice		Bob
1.	Choose random bit $b_A \in \{0,1\}$	$\xrightarrow{b_A}$.	
2.		$\xleftarrow{b_B}$	Choose random bit $b_B \in \{0,1\}$.
3.	Coin outcome is $b = b_A \oplus b_B$.		Coin outcome is $b = b_A \oplus b_B$.

Alice considers this for awhile, then objects.

"This isn't fair. You get to see my coin before I see yours, so now you have complete control over the outcome."

Alice's counter proposal

She suggests that she would be happy if the first two steps were reversed, so that Bob flips his coin first, but Bob balks at that suggestion.

They then both remember about blobs and decide to use blobs to prevent either party from controlling the outcome. They agree on the following protocol.

A mutually acceptable protocol

Alice		Bob
1. Choose random key k_A. Choose random bit $b_A \in \{0,1\}$.		Choose random key k_B. Choose random bit $b_B \in \{0,1\}$.
2. Compute $c_A = \text{commit}_{k_A}(b_A)$.	$\xleftrightarrow{c_A,\ c_B}$	Compute $c_B = \text{commit}_{k_B}(b_B)$.
3. Send k_A.	$\xleftrightarrow{k_A,\ k_B}$	Send k_B.
4. $b_B = \text{open}_{k_B}(c_B)$. Coin outcome is $b = b_A \oplus b_B$.		$b_A = \text{open}_{k_A}(c_A)$. Coin outcome is $b = b_A \oplus b_B$.

At the completion of step 2, both Alice and Bob have each others' commitment (something they failed to achieve in the past, which is why they're in the middle of a divorce now), but neither knows the other's private bit.

They learn each other's bit at the completion of steps 3 and 4.

Remaining asymmetry

While this protocol appears to be completely symmetric, it really isn't quite, for one of the parties completes step 3 before the other one does.

Say Alice receives s_B before sending s_A.

At that point, she can compute b_B and hence know the coin outcome b.

If it turns out that she lost, she might decide to stop the protocol and refuse to complete her part of step 3.

Premature termination

What happens if one party quits in the middle or detects the other party cheating?

So far, we've only considered the possibility of undetected cheating.

But in any real situation, one party might feel that he or she stands to gain by cheating, *even if the cheating is detected*.

Responses to cheating

Detected cheating raises complicated questions as to what happens next.

- Does a third party Carol become involved?
- If so, can Bob prove to Carol that Alice cheated?
- What if Alice refuses to talk to Carol?

Think about Bob's recourse in similar real-life situations and consider the reasons why such situations rarely arise.

For example, what happens if someone

- fails to follow the provisions of a contract?
- ignores a summons to appear in court?

A copycat attack

There is a subtle problem with the previous coin-flipping protocol.

Suppose Bob sends his message before Alice sends hers in each of steps 2 and 3.

Then Alice can choose $k_A = k_B$ and $c_A = c_B$ rather than following her proper protocol, so

$$\text{open}_{k_A}(c_A) = \text{open}_{k_B}(c_B).$$

In step 4, Bob will compute $b_A = b_B$ and won't detect that anything is wrong. The coin outcome is $b = b_A \oplus b_B = 0$. Hence, Alice can force outcome 0 simply by playing copycat.

Copycat attacks are not so easy to prevent.

Appendix: Security of PRSG Bit Commitment

Cryptographically strong PRSG looks random

Assuming G is cryptographically strong, then c will look random to Bob, regardless of the value of b, so he will be unable to get any information about b.

Why?

Assume Bob has advantage ϵ at guessing b when he can choose r and is given c. Here's a judge J for distinguishing $G(S)$ from U.

- Given input y, J chooses random b and simulates Bob's cheating algorithm. J simulates Bob choosing r, computes $c = y \oplus r^b$, and continues Bob's algorithm to find a guess $\hat{b}$ for b.
- If $\hat{b} = b$, J outputs 1.
- If $\hat{b} \neq b$, J outputs 0.

The judge's advantage

If y is drawn at random from U, then c is uniformly distributed and independent of b, so J outputs 1 with probability $1/2$.

If y comes from $G(S)$, then J outputs 1 with the same probability that Bob can correctly guess b.

Assuming G is cryptographically strong, then Bob has negligible advantage at guessing b.

Purpose of r

The purpose of r is to protect Bob against a cheating Alice.

Alice can cheat if she can find a triple (c, s_0, s_1) such that s_0 opens c to reveal 0 and s_1 opens c to reveal 1.

Such a triple must satisfy the following pair of equations:

$$\left.\begin{array}{rcl} c & = & G_\rho(s_0) \\ c & = & G_\rho(s_1) \oplus r. \end{array}\right\}$$

It is sufficient for her to solve the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1)$$

for s_0 and s_1 and then choose $c = G_\rho(s_0)$.

How big does ρ need to be?

We now count the number of values of r for which the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1)$$

has a solution.

Suppose n is the seed length, so the number of seeds is $\leq 2^n$. Then the right side of the equation can assume at most $2^{2n}/2$ distinct values.

Among the 2^ρ possible values for r, only 2^{2n-1} of them have the possibility of a colliding triple, regardless of whether or not Alice can feasibly find it.

Hence, by choosing ρ sufficiently much larger than $2n-1$, the probability of Alice cheating can be made arbitrarily small.

For example, if $\rho = 2n + 19$ then her probability of successful cheating is at most 2^{-20}.

Why does Bob need to choose r?

Why can't Alice choose r, or why can't r be fixed to some constant?

If Alice chooses r, then she can easily solve $r = G_\rho(s_0) \oplus G_\rho(s_1)$ and cheat.

If r is fixed to a constant, then if Alice ever finds a colliding triple (c, s_0, s_1), she can fool Bob every time.

While finding such a pair would be difficult if G_ρ were a truly random function, any specific PRSG might have special properties, at least for a few seeds, that would make this possible.

Example

For example, suppose $r = 1^\rho$ and $G_\rho(\neg s_0) = \neg G_\rho(s_0)$ for some s_0.

Then taking $s_1 = \neg s_0$ gives
$G_\rho(s_0) \oplus G_\rho(s_1) = G_\rho(s_0) \oplus G_\rho(\neg s_0) = G_\rho(s_0) \oplus \neg G_\rho(s_0) = 1^\rho = r$.

By having Bob choose r at random, r will be different each time (with very high probability).

A successful cheating Alice would be forced to solve
$r = G_\rho(s_0) \oplus G_\rho(s_1)$ in general, not just for one special case.