The Legendre and Jacobi Symbols
- The Legendre symbol
- Jacobi symbol
- Computing the Jacobi symbol

BBS Pseudorandom Sequence Generator

Bit Commitment Problem
- Bit Commitment Using QR Cryptosystem
- Bit Commitment Using Symmetric Cryptography
- Bit Commitment Using Hash Functions
- Bit Commitment Using Pseudorandom Sequence Generators

Appendix 1: Formalization of Bit Commitment Schemes

Appendix 2: Security of BBS
The Legendre and Jacobi Symbols
Notation for quadratic residues

The Legendre and Jacobi symbols form a kind of calculus for reasoning about quadratic residues and non-residues. They lead to a feasible algorithm for determining membership in $Q_n^{01} \cup Q_n^{10}$. Like the Euclidean gcd algorithm, the algorithm does not require factorization of its arguments.

The existence of this algorithm also explains why the Goldwasser-Micali cryptosystem can’t use all of $\mathbb{Q}_{n}^\text{NQR}$ in the encryption of “1”, for those elements in $Q_n^{01} \cup Q_n^{10}$ are readily determined to be in $\mathbb{Q}_{n}^\text{NQR}$. 
Legendre symbol

Let $p$ be an odd prime, $a$ an integer. The Legendre symbol $\left( \frac{a}{p} \right)$ is a number in $\{-1, 0, +1\}$, defined as follows:

$$\left( \frac{a}{p} \right) = \begin{cases} 
+1 & \text{if } a \text{ is a non-trivial quadratic residue modulo } p \\
0 & \text{if } a \equiv 0 \pmod{p} \\
-1 & \text{if } a \text{ is not a quadratic residue modulo } p 
\end{cases}$$

By the Euler Criterion, we have

**Theorem**

Let $p$ be an odd prime. Then

$$\left( \frac{a}{p} \right) \equiv a^{\left( \frac{p-1}{2} \right)} \pmod{p}$$

Note that this theorem holds even when $p | a$. 
Properties of the Legendre symbol

The Legendre symbol satisfies the following *multiplicative property*:

**Fact**

*Let* $p$ *be an odd prime. Then*

$$\left( \frac{a_1 a_2}{p} \right) = \left( \frac{a_1}{p} \right) \left( \frac{a_2}{p} \right)$$

Not surprisingly, if $a_1$ and $a_2$ are both non-trivial quadratic residues, then so is $a_1 a_2$. Hence, the fact holds when

$$\left( \frac{a_1}{p} \right) = \left( \frac{a_2}{p} \right) = 1.$$
Product of two non-residues

Suppose $a_1 \not\in \mathbb{QR}_p$, $a_2 \not\in \mathbb{QR}_p$. The above fact asserts that the product $a_1 a_2$ is a quadratic residue since

$$\left( \frac{a_1 a_2}{p} \right) = \left( \frac{a_1}{p} \right) \left( \frac{a_2}{p} \right) = (-1)(-1) = 1.$$  

Here’s why.

- Let $g$ be a primitive root of $p$.
- Write $a_1 \equiv g^{k_1} \pmod{p}$ and $a_2 \equiv g^{k_2} \pmod{p}$.
- Both $k_1$ and $k_2$ are odd since $a_1$, $a_2 \not\in \mathbb{QR}_p$.
- But then $k_1 + k_2$ is even.
- Hence, $g^{(k_1+k_2)/2}$ is a square root of $a_1 a_2 \equiv g^{k_1+k_2} \pmod{p}$, so $a_1 a_2$ is a quadratic residue.
The Jacobi symbol

The Jacobi symbol extends the Legendre symbol to the case where the “denominator” is an arbitrary odd positive number $n$.

Let $n$ be an odd positive integer with prime factorization $\prod_{i=1}^{k} p_i^{e_i}$. We define the Jacobi symbol by

$$\left( \frac{a}{n} \right) = \prod_{i=1}^{k} \left( \frac{a}{p_i} \right)^{e_i}$$

(The symbol on the left is the Jacobi symbol, and the symbol on the right is the Legendre symbol.)

(By convention, this product is 1 when $k = 0$, so $\left( \frac{a}{1} \right) = 1$.)

The Jacobi symbol extends the Legendre symbol since the two definitions coincide when $n$ is an odd prime.
What does the Jacobi symbol mean when $n$ is not prime?

- If $(\frac{a}{n}) = +1$, $a$ might or might not be a quadratic residue.
- If $(\frac{a}{n}) = 0$, then $\gcd(a, n) \neq 1$.
- If $(\frac{a}{n}) = -1$ then $a$ is definitely not a quadratic residue.
Jacobi symbol $= +1$ for $n = pq$

Let $n = pq$ for $p, q$ distinct odd primes. Since

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

there are two cases that result in $\left( \frac{a}{n} \right) = 1$:

1. $\left( \frac{a}{p} \right) = \left( \frac{a}{q} \right) = +1$, or

2. $\left( \frac{a}{p} \right) = \left( \frac{a}{q} \right) = -1$. 
Case of both Jacobi symbols $= +1$

If $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = +1$, then $a \in \mathbb{QR}_p \cap \mathbb{QR}_q = Q_{11}$.

It follows by the Chinese Remainder Theorem that $a \in \mathbb{QR}_n$.

This fact was implicitly used in the proof sketch that $|\sqrt{a}| = 4$. 
Case of both Jacobi symbols $= -1$

If $\left( \frac{a}{p} \right) = \left( \frac{a}{q} \right) = -1$, then $a \in \mathbb{QNR}_p \cap \mathbb{QNR}_q = Q_n^{00}$.

In this case, $a$ is not a quadratic residue modulo $n$.

Such numbers $a$ are sometimes called “pseudo-squares” since they have Jacobi symbol 1 but are not quadratic residues.
Computing the Jacobi symbol

The Jacobi symbol \( \left( \frac{a}{n} \right) \) is easily computed from its definition (equation 1) and the Euler Criterion, given the factorization of \( n \).

Similarly, \( \gcd(u, v) \) is easily computed without resort to the Euclidean algorithm given the factorizations of \( u \) and \( v \).

The remarkable fact about the Euclidean algorithm is that it lets us compute \( \gcd(u, v) \) efficiently, without knowing the factors of \( u \) and \( v \).

A similar algorithm allows us to compute the Jacobi symbol \( \left( \frac{a}{n} \right) \) efficiently, without knowing the factorization of \( a \) or \( n \).
Identities involving the Jacobi symbol

The algorithm is based on identities satisfied by the Jacobi symbol:

1. \( \left( \frac{0}{n} \right) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n \neq 1 \end{cases} \)

2. \( \left( \frac{2}{n} \right) = \begin{cases} 1 & \text{if } n \equiv \pm 1 \pmod{8} \\ -1 & \text{if } n \equiv \pm 3 \pmod{8} \end{cases} \)

3. \( \left( \frac{a_1}{n} \right) = \left( \frac{a_2}{n} \right) \text{ if } a_1 \equiv a_2 \pmod{n} \)

4. \( \left( \frac{2a}{n} \right) = \left( \frac{2}{n} \right) \cdot \left( \frac{a}{n} \right) \)

5. \( \left( \frac{a}{n} \right) = \begin{cases} \left( \frac{n}{a} \right) & \text{if } a, n \text{ odd and } \neg(a \equiv n \equiv 3 \pmod{4}) \\ -\left( \frac{n}{a} \right) & \text{if } a, n \text{ odd and } a \equiv n \equiv 3 \pmod{4} \end{cases} \)
A recursive algorithm for computing Jacobi symbol

/* Precondition: a, n >= 0; n is odd */
int jacobi(int a, int n) {
    if (a == 0) /* identity 1 */
        return (n==1) ? 1 : 0;
    if (a == 2) /* identity 2 */
        switch (n%8) {
            case 1: case 7: return 1;
            case 3: case 5: return -1;
        }
    if ( a >= n ) /* identity 3 */
        return jacobi(a%n, n);
    if (a%2 == 0) /* identity 4 */
        return jacobi(2,n)*jacobi(a/2, n);
    /* a is odd */ /* identity 5 */
    return (a%4 == 3 && n%4 == 3) ? -jacobi(n,a) : jacobi(n,a);
}
BBS Pseudorandom Sequence Generator
Blum primes and integers

A **Blum prime** is a prime $p$ such that $p \equiv 3 \pmod{4}$.

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

If $p$ is a Blum prime, then $-1 \in \mathbb{QNR}_p$. This follows from the Euler criterion, since $\frac{p-1}{2}$ is odd. By definition of the Legendre symbol, $\left(\frac{-1}{p}\right) = -1$.

If $n$ is a Blum integer, then $-1 \in \mathbb{QNR}_n$, but now

$$\left(\frac{-1}{n}\right) = \left(\frac{-1}{p}\right) \left(\frac{-1}{q}\right) = (-1)(-1) = 1.$$
Square roots of Blum primes

Theorem
Let \( p \) be a Blum prime, \( a \in \text{QR}_p \), and \( \{b, -b\} = \sqrt{a} \) be the two square roots of \( a \). Then exactly one of \( b \) and \(-b\) is itself a quadratic residue.

Proof. 
\((-b)^{(p-1)/2} \neq b^{(p-1)/2}\) since 
\[(-b)^{(p-1)/2} = (-1)^{(p-1)/2} b^{(p-1)/2} = (-1)b^{(p-1)/2}.\]
Both \((-b)^{(p-1)/2}\) and \(b^{(p-1)/2}\) are in \(\sqrt{1} = \{\pm 1\}\), so it follows from the Euler criterion that one of \(b, -b\) is a quadratic residue and the other is not.
Square roots of Blum integers

**Theorem (QR square root)**

Let \( n = pq \) be a Blum integer and \( a \in \mathbb{QR}_n \). **Exactly one of a’s four square roots modulo \( n \) is a quadratic residue.**
Proof of QR square root theorem

Consider $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$. $a \in \text{QR}_p$ and $a \in \text{QR}_q$.

Let $\{b, -b\} \in \sqrt{a} \pmod{p}$. By the previous theorem, exactly one of these numbers is in $\text{QR}_p$. Call that number $b_p$.

Similarly, one of the square roots of $a \pmod{q}$ is in $\text{QR}_q$, say $b_q$.

Applying the Chinese Remainder Theorem, it follows that exactly one of $a$’s four square roots modulo $n$ is in $\text{QR}_n$. 
A cryptographically secure PRSG

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

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

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

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

The $\ell$-bit output sequence $\text{BBS}(s_0)$ is $b_1, b_2, b_3, \ldots, b_\ell$, where $b_i = \text{lsb}(s_i)$ is the least significant bit of $s_i$. 
The security of BBS is based on the assumed difficulty of determining, for a given $a$ with Jacobi symbol 1, whether or not $a$ is a quadratic residue, i.e., whether or not $a \in \mathbb{QR}_n$.

We just showed that 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 (ambiguously) by $\sqrt{a} \pmod{n}$ or simply by $\sqrt{a}$ when it is clear that mod $n$ is intended.
Security of BBS

We show in appendix 1 that BBS is cryptographically secure.

The proof reduces the problem of predicting the output of BBS to the quadratic residuosity problem for numbers with Jacobi symbol 1 over Blum integers.

To do this reduction, we show that if there is a judge $J$ that successfully distinguishes $\text{BBS}(S)$ from $U$, then there is a feasible algorithm $A$ for distinguishing quadratic residues from non-residues with Jacobi symbol 1, contradicting the above version of the QR hardness assumption.
Bit Commitment Problem
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.
Bit commitment

A *bit-commitment* is an encryption of a bit using a cryptosystem with a special property.

1. The bit is hidden from anyone not knowing the secret key.
2. There is only one valid way of decrypting the ciphertext, no matter what key is used.

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

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


Bit commitment intuition

In other words,

- If Alice produces a commitment $c$ to a bit $b$, then $b$ cannot be recovered from $c$ without knowing Alice’s secret encoding key $k$.
- There is no key $k'$ that Alice might release that would make it appear that $c$ is a commitment of the bit $1 - b$. 
Bit-commitments as cryptographic envelopes

More formally, a *bit commitment* or *blob* or *cryptographic envelope* is an electronic analog of a sealed envelope.

Intuitively, a blob has two properties:

1. The bit inside the blob *remains hidden* until the blob is opened.
2. The bit inside the blob *cannot be changed*, that is, the blob cannot be opened in different ways to reveal different bits.
Bit-commitment primitives

A blob is produced by a protocol \texttt{commit}(b) between Alice and Bob. We assume initially that only Alice knows \( b \).

At the end of the commit protocol, Bob has a blob \( c \) containing Alice’s bit \( b \), but he should have no information about \( b \)’s value.

Later, Alice and Bob can run a protocol \texttt{open}(c) to reveal the bit contained in \( c \) to Bob.
Requirements for bit 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 $b$ after she runs $\text{commit}(b)$, even if he cheats.

▶ Bob wants to be sure that all successful runs of $\text{open}(c)$ reveal the same bit $b'$, no matter what Alice does.

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

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 bit $b$.

1. Create a Goldwasser-Micali public key $e = (n, y)$, where $n = pq$.
2. Choose random $r \in \mathbb{Z}_n^*$, and use it to produce an encryption $c$ of $b$. (See lecture 19, slide 29.)
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 encryption of $b$ based on $r$. 
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 Symmetric Cryptography
A naïve approach for a faster bit-commitment scheme

A naïve way to use a symmetric cryptosystem for bit commitment is for Alice to encrypt $b$ with a private key $k$ to get blob $c = E_k(b)$. She opens it by releasing $k$. Anyone can compute $b = D_k(c)$.

Alice can easily cheat if she can find a colliding triple $(c, k_0, k_1)$ with the property that $D_{k_0}(c) = 0$ and $D_{k_1}(c) = 1$. She “commits” by sending $c$ to Bob.

Later, she can choose to send Bob either $k_0$ or $k_1$. This isn’t just a hypothetical problem. Suppose Alice uses the most secure cryptosystem of all, a one-time pad, so $D_k(c) = c \oplus k$.

Then $(c, c \oplus 0, c \oplus 1)$ is a colliding triple.
Another attempt

The protocol below tries to make it harder for Alice to cheat by making it possible for Bob to detect most bad keys.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To commit</strong>(b):</td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Choose random string ( r ).</td>
</tr>
<tr>
<td>2. Choose random key ( k ).</td>
<td>Compute ( c = E_k(r \cdot b) ).</td>
</tr>
<tr>
<td>Compute ( c = E_k(r \cdot b) ).</td>
<td>( c ) is commitment.</td>
</tr>
<tr>
<td><strong>To open</strong>(c):</td>
<td></td>
</tr>
<tr>
<td>3. Send ( k ).</td>
<td>Let ( r' \cdot b' = D_k(c) ).</td>
</tr>
<tr>
<td></td>
<td>Check ( r' = r ).</td>
</tr>
<tr>
<td></td>
<td>( b' ) is revealed bit.</td>
</tr>
</tbody>
</table>
Security of second attempt

For many cryptosystems (e.g., DES), this protocol does indeed prevent Alice from cheating, for she will have difficulty finding any two keys $k_0$ and $k_1$ such that $E_{k_0}(r \cdot 0) = E_{k_1}(r \cdot 1)$, and $r$ is different for each run of the protocol.

However, for the one-time pad, she can cheat as before: She just takes $c$ to be random and lets $k_0 = c \oplus (r \cdot 0)$ and $k_1 = c \oplus (r \cdot 1)$.

Then $D_{k_b}(c) = r \cdot b$ for $b \in \{0, 1\}$, so the revealed bit is 0 or 1 depending on whether Alice sends $k_0$ or $k_1$ in step 3.
Need for a different approach

We see that not all secure cryptosystems have the properties we need in order to make the protocol secure.

We need a property analogous to the strong collision-free property for hash functions (lecture 14).
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. \( r_1 \leftarrow \) Choose random string \( r_1 \).
2. Choose random string \( r_2 \).
3. Send \( r_2 \).

\[ c = H(r_1 r_2 b) \]

\( c \rightarrow \) is commitment.

To open(c):

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 $H(r_20) = c = H(r'_21)$.

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
Bit commitment using a PRSG

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

Alice

To commit($b$):

1. 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 $c = y \oplus r$.

Bob

$c$ is commitment.

To open($c$):

3. Send $s$.
   Let $y = G_\rho(s)$.
   If $c = y$ then reveal 0.
   If $c = y \oplus r$ then reveal 1.
   Otherwise, fail.
Security of PRSG bit commitment

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 $\epsilon$ 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:

\[
\begin{align*}
    c &= G_\rho(s_0) \\
    c &= G_\rho(s_1) \oplus r.
\end{align*}
\]

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 $\rho$ 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 $2^\rho$ 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 $\rho$ 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 $G_\rho$ 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.
Appendix 1: Formalization of Bit Commitment Schemes
Formalization of bit commitment schemes

The above bit commitment protocols all have the same form.

We abstract from them a cryptographic primitive, called a \textit{bit commitment scheme}, which consists of a pair of \textit{key spaces} $\mathcal{K}_A$ and $\mathcal{K}_B$, a \textit{blob space} $\mathcal{B}$, a \textit{commitment} function

$$\text{enclose} : \mathcal{K}_A \times \mathcal{K}_B \times \{0, 1\} \rightarrow \mathcal{B},$$

and an \textit{opening} function

$$\text{reveal} : \mathcal{K}_A \times \mathcal{K}_B \times \mathcal{B} \rightarrow \{0, 1, \phi\},$$

where $\phi$ means “failure”.

We say that a blob $c \in \mathcal{B}$ \textit{contains} $b \in \{0, 1\}$ if

$$\text{reveal}(k_A, k_B, c) = b \text{ for some } k_A \in \mathcal{K}_A \text{ and } k_B \in \mathcal{K}_B.$$
 Desired properties

These functions have three properties:

1. \( \forall k_A \in \mathcal{K}_A, \forall k_B \in \mathcal{K}_B, \forall b \in \{0, 1\}, \)
   \( \text{reveal}(k_A, k_B, \text{enclose}(k_A, k_B, b)) = b \);

2. \( \forall k_B \in \mathcal{K}_B, \forall c \in \mathcal{B}, \exists b \in \{0, 1\}, \forall k_A \in \mathcal{K}_A, \)
   \( \text{reveal}(k_A, k_B, c) \in \{b, \phi\} \).

3. No feasible probabilistic algorithm that attempts to distinguish
   blobs containing 0 from those containing 1, given \( k_B \) and \( c \), is
   correct with probability significantly greater than 1/2.
Intuition

The intention is that $k_A$ is chosen by Alice and $k_B$ by Bob. Intuitively, these conditions say:

1. Any bit $b$ can be committed using any key pair $k_A, k_B$, and the same key pair will open the blob to reveal $b$.
2. For each $k_B$, all $k_A$ that successfully open $c$ reveal the same bit.
3. Without knowing $k_A$, the blob does not reveal any significant amount of information about the bit it contains, even when $k_B$ is known.
Comparison with symmetric cryptosystem

A bit commitment scheme looks a lot like a symmetric cryptosystem, with \texttt{enclose}(k_A, k_B, b) playing the role of the encryption function and \texttt{reveal}(k_A, k_B, c) the role of the decryption function.

However, they differ both in their properties and in the environments in which they are used.

Conventional cryptosystems do not require uniqueness condition 2, nor do they necessarily satisfy it.
Comparison with symmetric cryptosystem (cont.)

In a conventional cryptosystem, we assume that Alice and Bob trust each other and both share a secret key $k$.

The cryptosystem is designed to protect Alice’s secret message from a passive eavesdropper Eve.

In a bit commitment scheme, Alice and Bob cooperate in the protocol but do not trust each other to choose the key.

Rather, the key is split into two pieces, $k_A$ and $k_B$, with each participant controlling one piece.
A bit-commitment protocol from a bit-commitment scheme

A bit commitment scheme can be turned into a bit commitment protocol by plugging it into the generic protocol:

\[
\begin{align*}
\text{Alice} & \quad \text{Bob} \\
\text{To commit}(b): & \\
1. & \quad \begin{aligned} & k_B \quad \text{Choose random } k_B \in \mathcal{K}_B. \\ & k_A \quad \text{Choose random } k_A \in \mathcal{K}_A. \\ & c = \text{enclose}(k_A, k_B, b). \quad c \quad \text{is commitment.} \end{aligned} \\
2. & \quad \text{Compute } b = \text{reveal}(k_A, k_B, c). \quad \text{If } b = \phi, \text{ then fail.} \\
3. & \quad \text{If } b \neq \phi, \text{ then } b \text{ is revealed bit.}
\end{align*}
\]
The previous bit commitment protocols we have presented can all be regarded as instances of the generic protocol.

For example, we get the second protocol based on symmetric cryptography by taking

\[
\text{enclose}(k_A, k_B, b) = E_{k_A}(k_B \cdot b),
\]

and

\[
\text{reveal}(k_A, k_B, c) = \begin{cases} 
  b & \text{if } k_B \cdot b = D_{k_A}(c) \\
  \phi & \text{otherwise.}
\end{cases}
\]
Appendix 2: Security of BBS
Blum integers and the Jacobi symbol

Fact

Let \( n \) be a Blum integer and \( a \in \mathbb{QR}_n \). 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,

\[
\left( \frac{a}{p} \right) = -\left( \frac{-a}{p} \right) \quad \text{and} \quad \left( \frac{a}{q} \right) = -\left( \frac{-a}{q} \right),
\]

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 $\text{lsb}(x) = (x \mod 2)$ be the least significant bit of integer $x$.

**Fact**

*If $n$ is odd, then $\text{lsb}(x \mod n) \oplus \text{lsb}((-x) \mod n) = 1$.***
First-bit prediction

A *first-bit predictor with advantage* $\epsilon$ 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 $\text{BBS}(S)$ 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 $\text{BBS}(S)$ 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 $\epsilon$, 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 $\epsilon$. 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 $x \in \mathbb{QR}_n$ and 0 to mean $x \notin \mathbb{QR}_n$.

To $Q(x)$:
1. Let $\hat{s}_2 = x^2 \mod n$.
2. Let $\hat{s}_i = \hat{s}_{i-1}^2 \mod n$, for $i = 3, \ldots, \ell$.
3. Let $\hat{b}_1 = \text{lsb}(x)$.
4. Let $\hat{b}_i = \text{lsb}(\hat{s}_i)$, 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 $\text{BBS}(s_0)$.

We have two cases.

Case 1: $x \in \mathbb{QR}_n$. Then $s_1 = x$, so the state sequence of $\text{BBS}(s_0)$ 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: $x \in \text{QNR}_n$, so $-x \in \text{QR}_n$. Then $s_1 = -x$, so the state sequence of $\text{BBS}(s_0)$ 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 $1/2 + \epsilon$. 