CPSC 367: Cryptography and Security

Michael J. Fischer

Lecture 21
April 11, 2019

Quadratic Residues Revisited

QR reprise

Quadratic residues play a key role in the Feige-Fiat-Shamir zero knowledge authentication protocol.

They can also be used to produce a secure probabilistic cryptosystem and a cryptographically strong pseudorandom bit generator.

Before we can proceed to these protocols, we need some more number-theoretic properties of quadratic residues.

Euler criterion

The Euler criterion gives a feasible method for testing membership in QR_p when p is an odd prime.

Theorem (Euler Criterion)

An integer a is a non-trivial[1] *quadratic residue modulo an odd prime p iff*

$$a^{(p-1)/2} \equiv 1 \pmod{p}.$$

[1] A non-trivial quadratic residue is one that is not equivalent to $0 \pmod{p}$.

Proof of Euler Criterion

Proof in forward direction.

Let $a \equiv b^2 \pmod{p}$ for some $b \not\equiv 0 \pmod{p}$. Then

$$a^{(p-1)/2} \equiv (b^2)^{(p-1)/2} \equiv b^{p-1} \equiv 1 \pmod{p}$$

by Euler's theorem, as desired. $\square$

Proof of Euler Criterion (continued)

Proof in reverse direction.

Suppose $a^{(p-1)/2} \equiv 1 \pmod{p}$. Clearly $a \not\equiv 0 \pmod{p}$. We find a square root b of a modulo p.

Let g be a primitive root of p. Choose k so that $a \equiv g^k \pmod{p}$, and let $\ell = (p-1)k/2$. Then

$$g^{\ell} \equiv g^{(p-1)k/2} \equiv (g^k)^{(p-1)/2} \equiv a^{(p-1)/2} \equiv 1 \pmod{p}.$$

Since g is a primitive root and $g^{\ell} \equiv 1 \pmod{p}$, then $\phi(p) | \ell$. Hence, $\ell/\phi(p) = \ell/(p-1) = k/2$ is an integer.

Let $b = g^{k/2}$. Then $b^2 \equiv g^k \equiv a \pmod{p}$, so b is a non-trivial square root of a modulo p, as desired. $\square$

A hard problem associated with quadratic residues

Let $n = pq$, where p and q are distinct odd primes.

Recall that each $a \in \mathrm{QR}_n$ has 4 square roots, and $1/4$ of the elements in $\mathbf{Z}_n^*$ are quadratic residues.

Some elements of $\mathbf{Z}_n^*$ are easily recognized as non-residues, but there is a subset of non-residues (which we denote by Q_n^{00}) that are *hard to distinguish* from quadratic residues without knowing p and q.

Quadratic residues modulo $n = pq$

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

We divide the numbers in $\mathbf{Z}_n^*$ into four classes depending on their membership in QR_p and QR_q.[2]

- Let $Q_n^{11} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QR}_p \cap \mathrm{QR}_q\}$.
- Let $Q_n^{10} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QR}_p \cap \mathrm{QNR}_q\}$.
- Let $Q_n^{01} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QNR}_p \cap \mathrm{QR}_q\}$.
- Let $Q_n^{00} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QNR}_p \cap \mathrm{QNR}_q\}$.

Under these definitions, $$\mathrm{QR}_n = Q_n^{11}$$

$$\mathrm{QNR}_n = Q_n^{00} \cup Q_n^{01} \cup Q_n^{10}$$

[2] To be strictly formal, we classify $a \in \mathbf{Z}_n^*$ according to whether or not $(a \bmod p) \in \mathrm{QR}_p$ and whether or not $(a \bmod q) \in \mathrm{QR}_q$.

Quadratic residuosity problem

Definition (Quadratic residuosity problem)

The *quadratic residuosity problem* is to decide, given $a \in Q_n^{00} \cup Q_n^{11}$, whether or not $a \in Q_n^{11}$.

Fact

There is no known feasible algorithm for solving the quadratic residuosity problem that gives the correct answer significantly more than 1/2 the time for uniformly distributed random $a \in Q_n^{00} \cup Q_n^{11}$*, unless the factorization of n is known.*

The *quadratic residuosity assumption* is that there is not feasible algorithm for solving the quadradic residuosity problem that gives the correct answer with probability significantly better than $1/2$.

Encryption Based on Quadratic Residues

Securely encrypting single bits

Goldwasser and Micali devised a probabilistic public key cryptosystem based on the assumed hardness of the quadratic residuosity problem that allows one to securely encrypt single bits.

The idea is to encrypt a "0" by a random residue of QR_n and a "1" by a random non-residue in Q_n^{00}. Any ability to decrypt the bit is tantamount to solving the quadratic residuosity problem.

Goldwasser-Micali probabilistic cryptosystem

Key Generation

The public key consists of a pair $e = (n, y)$, where $n = pq$ for distinct odd primes p, q, and y is any member of Q_n^{00}.

The private key consists of the triple $d = (n, y, p)$.

The message space is $\mathcal{M} = \{0, 1\}$. (Single bits!)

The ciphertext space is $\mathcal{C} = Q_n^{00} \cup Q_n^{11}$.

Goldwasser-Micali probabilistic cryptosystem (cont.)

Encryption

To encrypt $m \in \mathcal{M}$, Alice chooses a random $r \in \mathbf{Z}_n^*$ and sets $a = r^2 \bmod n$. The result a is a random element of $\mathrm{QR}_n = Q_n^{11}$.

> If $m = 0$, set $c = a$ (which is in Q_n^{11}).
> If $m = 1$, set $c = ay \bmod n$ (which is in Q_n^{00}).

Decryption

Bob, knowing the private key p, can use the Euler Criterion to quickly determine whether or not $c \in \mathrm{QR}_p$ and hence whether $c \in Q_n^{11}$ or $c \in Q_n^{00}$, thereby determining m.

Goldwasser-Micali probabilistic cryptosystem (cont.)

Security

Eve's problem of finding m given c is equivalent to the problem of testing if $c \in Q_n^{11}$, given that $c \in Q_n^{00} \cup Q_n^{11}$.

This is just the quadratic residuosity problem, assuming the ciphertexts are uniformly distributed. One can show:

- Every element of Q_n^{11} is equally likely to be chosen as the ciphertext c in case $m = 0$;
- Every element of Q_n^{00} is equally likely to be chosen as the ciphertext c in case $m = 1$.

If the messages are also uniformly distributed, then any element of $Q_n^{00} \cup Q_n^{11}$ is equally likely to be the ciphertext.

Important facts about quadratic residues

1. If p is odd prime, then $|\mathrm{QR}_p| = |\mathbf{Z}_p^*|/2$, and for each $a \in \mathrm{QR}_p$, $|\sqrt{a}| = 2$.
2. If $n = pq$, $p \neq q$ odd primes, then $|\mathrm{QR}_n| = |\mathbf{Z}_n^*|/4$, and for each $a \in \mathrm{QR}_n$, $|\sqrt{a}| = 4$.
3. Euler criterion: $a \in \mathrm{QR}_p$ iff $a^{(p-1)/2} \equiv 1 \pmod{p}$, p odd prime.
4. If p is odd prime, $a \in \mathrm{QR}_p$, can feasibly find $y \in \sqrt{a}$. (See appendix.)
5. If $n = pq$, $p \neq q$ odd primes, then distinguishing Q_n^{00} from Q_n^{11} is believed to be infeasible. Hence, infeasible to find $y \in \sqrt{a}$. Why?
   If not, one could attempt to find $y \in \sqrt{a}$, check that $y^2 \equiv a \pmod{n}$, and conclude that $a \in Q^{11}$ if successful.

Secure Random Sequence Generators

Pseudorandom 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

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

sets the 32-bit seed. Each subsequent call on

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

returns an integer in the range $[0, \ldots, RAND\_MAX]$.

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 $16 * (2^{31} - 1)$.

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,

The BBS secure PRSG

In the rest of this lecture, we show how to build a PRSG that is provably secure. It is based on the quadratic residuosity assumption.

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 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. By definition of the Legendre symbol, $\left(\frac{-1}{p}\right) = -1$. (See appendix.)

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

Square roots of Blum integers

Theorem (QR square root)

Let $n = pq$ be a Blum integer and $a \in \mathrm{QR}_n$. Exactly one of a's four square roots modulo n is a quadratic residue.

Proof of QR square root theorem

Consider $\mathbf{Z}_p^*$ and $\mathbf{Z}_q^*$. $a \in \mathrm{QR}_p$ and $a \in \mathrm{QR}_q$.

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

Similarly, one of the square roots of $a \pmod{q}$ is in 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 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 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.

QR assumption and Blum integers

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 $a \in \mathrm{QR}_n$. (See appendix.)

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 the appendix 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 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.

Appendix: Finding Square Roots

Finding square roots modulo prime $p \equiv 3 \pmod 4$

The Euler criterion lets us test membership in QR_p for prime p, but it doesn't tell us how to quickly find square roots. They are easily found in the special case when $p \equiv 3 \pmod 4$.

Theorem

Let $p \equiv 3 \pmod 4$, $a \in \mathrm{QR}_p$. Then $b = a^{(p+1)/4} \in \sqrt{a} \pmod p$.

Proof.

$p + 1$ is divisible by 4, so $(p+1)/4$ is an integer. Then

$$b^2 \equiv (a^{(p+1)/4})^2 \equiv a^{(p+1)/2} \equiv a^{1+(p-1)/2} \equiv a \cdot 1 \equiv a \pmod p$$

by the Euler Criterion. □

Finding square roots for general primes

We now present an algorithm due to D. Shanks[3] that finds square roots of quadratic residues modulo any odd prime p.

[3]Shanks's algorithm appeared in his paper, "Five number-theoretic algorithms", in Proceedings of the Second Manitoba Conference on Numerical Mathematics, Congressus Numerantium, No. VII, 1973, 51–70. Our treatment is taken from the paper by Jan-Christoph Schlage-Puchta", "On Shank's Algorithm for Modular Square Roots", *Applied Mathematics E-Notes, 5* (2005), 84–88.

Shank's algorithm

Let p be an odd prime. Write $\phi(p) = p - 1 = 2^s t$, where t is odd.
(Recall: s is # trailing 0's in the binary expansion of $p - 1$.)

Because p is odd, $p - 1$ is even, so $s \geq 1$.

A special case

In the special case when $s = 1$, then $p - 1 = 2t$, so $p = 2t + 1$.

Writing the odd number t as $2\ell + 1$ for some integer ℓ, we have

$$p = 2(2\ell + 1) + 1 = 4\ell + 3,$$

so $p \equiv 3 \pmod 4$.

This is exactly the case that we handled above.

Overall structure of Shank's algorithm

Let $p - 1 = 2^s t$ be as above, where p is an odd prime.

Assume $a \in \mathrm{QR}_p$ is a quadratic residue and $u \in \mathrm{QNR}_p$ is a quadratic non-residue.

We can easily find u by choosing random elements of $\mathbf{Z}_p^*$ and applying the Euler Criterion.

The goal is to find x such that $x^2 \equiv a \pmod{p}$.

Shanks's algorithm

Let s, t satisfy $p - 1 = 2^s t$ and t odd.
Let $u \in \mathrm{QNR}_p$.
$k = s$
$z = u^t \bmod p$
$x = a^{(t+1)/2} \bmod p$
$b = a^t \bmod p$
while $(b \not\equiv 1 \pmod{p})$ $\{$
    let m be the least integer with $b^{2^m} \equiv 1 \pmod{p}$
    $y = z^{2^{k-m-1}} \bmod p$
    $z = y^2 \bmod p$
    $b = bz \bmod p$
    $x = xy \bmod p$
    $k = m$
$\}$
return x

Loop invariant

The congruence

$$x^2 \equiv ab \pmod{p}$$

is easily shown to be a loop invariant.

It's clearly true initially since $x^2 \equiv a^{t+1}$ and $b \equiv a^t \pmod{p}$.

Each time through the loop, a is unchanged, b gets multiplied by y^2 (lines 10 and 11), and x gets multiplied by y (line 12); hence the invariant remains true regardless of the value of y.

If the program terminates, we have $b \equiv 1 \pmod{p}$, so $x^2 \equiv a$, and x is a square root of $a \pmod{p}$.

Termination proof (sketch)

The algorithm terminates after at most $s - 1$ iterations of the loop.

To see why, we look at the orders[4] of b and $z \pmod{p}$ and show the following loop invariant:

> *At the start of each loop iteration (before line 8),* $\mathrm{ord}(b)$ *is a power of 2 and* $\mathrm{ord}(b) < \mathrm{ord}(z) = 2^k$*.*

After line 8, $m < k$ since $2^m = \mathrm{ord}(b) < 2^k$. Line 13 sets $k = m$ for the next iteration, so k decreases on each iteration.

The loop terminates when $b \equiv 1 \pmod{p}$. Then $\mathrm{ord}(b) = 1 < 2^k$, so $k \geq 1$. Hence, the loop is executed at most $s - 1$ times.

[4] Recall that the order of an element g modulo p is the least positive integer k such that $g^k \equiv 1 \pmod{p}$.

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.

Appendix: Similarity of Probability Distributions

Formal definition of PRSG

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

We generally assume that all of the seeds in $\mathcal{S}$ 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 $\mathcal{S}$ of possible seeds.

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

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

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

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

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

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

U is what we usually mean by a *truly random* variable on ℓ-bit strings.

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 $0011111100001100110000\ldots$.

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 ℓ have the special form above?

Judges

Formally, a *judge* is a probabilistic polynomial-time algorithm J that takes an ℓ-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 $\mathrm{Pr}[\mathrm{X} = \mathrm{x}]$ of input x occurring.

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

$$\mathrm{Pr}[\mathrm{J}(\mathrm{X}) = 1] = \sum_{\mathrm{x} \in \mathcal{X}} \mathrm{Pr}[\mathrm{X} = \mathrm{x}] \cdot \mathrm{p}_{\mathrm{x}}.$$

Formal definition of indistinguishability

Two random variables X and Y are *ϵ-indistinguishable by judge* J if

$$|\mathrm{Pr}[\mathrm{J}(\mathrm{X}) = 1] - \mathrm{Pr}[\mathrm{J}(\mathrm{Y}) = 1]| < \epsilon.$$

Intuitively, we say that G is *cryptographically secure* if $G(S)$ and U are ϵ-indistinguishable for suitably small ϵ 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 ℓ, the error parameter ϵ, and the allowed running time of the judges

Further formal details may be found in Goldwasser and Bellare.

Appendix: 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 QNR_n in the encryption of "1", for those elements in $Q_n^{01} \cup Q_n^{10}$ are readily determined to be in QNR_n.

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:

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

By the Euler Criterion, we have

Theorem

Let p be an odd prime. Then

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

Note that this theorem holds even when $p \mid a$.

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 $a_1 \notin \mathrm{QR}_p$, $a_2 \notin \mathrm{QR}_p$. 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 $a_1, a_2 \notin \mathrm{QR}_p$.
- 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

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

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.

Meaning of Jacobi symbol

What does the Jacobi symbol mean when n is not prime?

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

Jacobi symbol $= +1$ for $n = pq$

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

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right) \tag{2}$$

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 $a \in \mathrm{QR}_p \cap \mathrm{QR}_q = Q_n^{11}$.

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

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 $a \in \mathrm{QNR}_p \cap \mathrm{QNR}_q = Q_n^{00}$.

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. $\left(\frac{a_1}{n}\right) = \left(\frac{a_2}{n}\right)$ 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);
}
```

Appendix: Security of BBS

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,

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

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

$\square$

Blum integers and the least significant bit

The low-order bits of $x \bmod n$ and $(-x) \bmod n$ 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 $\mathrm{BBS}(s_0)$ 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 $1/2 + \epsilon$.