Solution to the Millionaire's Problem

This is a solution to what is known as the millionaires' problem: two millionaires, Alice and Bob (it's always Alice and Bob!) want to know which is richer, without revealing their actual wealth. This is all based on work by a gentleman called Andrew Yao [1]; it is also a specific case of a general set of theorems.

To begin with, Alice and Bob need a public-key cryptographic system which is strong: in this example, we'll use RSA. Note that throughout this example * stands for multiplication and ^ stands for exponentiation (2^3 is 2 cubed).

Alice uses RSA, and has a public key which is (79, 3337). Her private key is 1019. To keep the example simple, Alice and Bob have worked out from their Rolls-Royce collections that they are both worth somewhere in the region of 1 to 10 million. They don't want to tell each other how rich they are, but they do want to know which is richer.

Alice is worth I millions, and Bob J millions. For this example we will say that Alice is worth 5 million (I = 5) and Bob 6 million (J = 6). We will just use the numbers 1 to 10, not the millions, although the technique is extendable to the millions.

Step 1

Bob picks a random N-bit integer called x (Alice will later use a N/2 bit prime, so the length of the integer is important - in the worked example the prime we will use is only roughly N/2 bits, but it works). We will let x = 1234 (4 decimal digits, N is about 14 bits). Bob first calculates C such that:
C = 1234^79 mod 3337 = 901 [C is, in fact, the RSA encipherment of x.]

Step 2

Bob takes C, and transmits to Alice C - J + 1: so 901 - 6 + 1 = 896. He sends 896 to Alice.

Step 3

Alice generates a series of numbers Y1, Y2, Y3 ... Y10 such that Y1 is the RSA decipherment of (C - J + 1); Y2 is the RSA decipherment of (C - J + 2); Y3 is the decipherment of (C + J + 3); ... ; Y10 is the RSA decipherment of (C - J + 10).

She can do this because although she does not know C or J, she does know (C - J + 1): it's the number Bob sent her. So we get the following table, where U goes from 1 to 10 and YU is Y1, Y2, Y3, Y4 ... Y10:

U	(C - J + U)	RSA function	YU
1	896	896^1019 mod 3337	1059
2	897	897^1019 mod 3337	1156
3	898	898^1019 mod 3337	2502
4	899	.	2918
5	900	.	385
6	901	.	1234 (as it should be)
7	902	.	296
8	903	.	1596
9	904	.	2804
10	905	905^1019 mod 3337	1311

Step 4

Alice now generates a random N/2 bit length prime p. In this case, we will use 107: this is 7 or 8 bits and so fits the requirement. Alice then generates Z1, Z2, Z3 ... Z10 by calculating Y1 mod p, Y2 mod p, Y3 mod p ... Y10 mod p. So for U = 1 to 10, and thus ZU = Z1 to Z10, we can complete the table:

U	(C - J + U)	RSA function	YU	ZU (= YU mod 107)
1	896	896^1019 mod 3337	1059	96
2	897	897^1019 mod 3337	1156	86
3	898	898^1019 mod 3337	2502	41
4	899	.	2918	29
5	900	.	385	64
6	901	.	1234	57
7	902	.	296	82
8	903	.	1596	98
9	904	.	2804	22
10	905	905^1019 mod 3337	1311	27

Note that p must be chosen so that all the ZUs differ by at least 2 (this can be non-trivial for smaller examples).

Step 5

This is the clever bit: Alice now transmits the prime p to Bob, and then sends 10 numbers. The first few numbers are Z1, Z2, Z3 ... up to the value of ZI, where I is Alice's wealth in millions (in this case 5). So Alice sends Z1, Z2, Z3, Z4 and Z5. The rest of the numbers she adds 1 to, so she sends Z6 + 1, Z7 + 1, Z8 + 1, Z9 + 1 and Z10 +1. The complete string she sends is:

p	Z1	Z2	Z3	Z4	Z5	Z6+1	Z7+1	Z8+1	Z9+1	Z10+1
107	96	86	41	29	64	58	83	99	23	28

Step 6

Bob now looks at the Jth number where J is his wealth in millions (6), excluding the first prime. He thus looks at 58. He also computes G = x mod p (x being his original random number and p being Alice's random prime which she transmitted to him). Thus:

G = 1234 mod 107 = 57.

Now, if the Jth number (58) is equal to G, then Alice is equal or greater to Bob in wealth (I >= J). If the Jth number is not equal to G, then Bob is wealthier than Alice (I < J). In our example, he is, as 57 does not equal 58.

Step 7

Bob tells Alice the result. I think you can see how it works: Alice starts adding 1 to numbers in the series greater than her value (5): Bob is checking to see if the one in his position in the series (6) has had one added to it: if it has, then he knows he must be wealthier than Alice.

This has all been done without either of them transmitting their wealth value; at the end they both know more: Alice knows Bob has 6 to 10 million, and Bob knows Alice has 1 to 5 million, but that is an inevitable conclusion of knowing which one is wealthier.

If the >= condition holds, they can run the protocol the other way round to see if Alice is actually richer, or if they both have the same wealth.

This technique is not cheat-proof (Bob could lie in step 7). Yao shows that such techniques can be constructed so that cheating can be limited, usually by employing extra steps.

Lastly, in the real world, you could use this technique: but it will take some processing power. If you wanted to cover the range 1 to 100,000,000, at a unit resolution, then Alice will be sending Bob a table of 100,000,000

numbers, which probably will be at least 8-byte integers. With realistic formatting overheads, this means that this table is on the order of a gigabyte. It's handleable, but the processing and storage implications of this technique in the real world is non-trivial.

References

A C Yao, Protocols for Secure Computations (extended abstract) Proceedings of the 21st Annual IEEE Symposium on the Foundations of Computer Science, 1982, pp 160-164.

Appendix: RSA

Produce a number n such that it is a product of two primes p and q:
n = p * q

Choose a number e so that it is relatively prime to (p - 1) * (q - 1). Note: the number (p - 1) * (q - 1) is the Euler totient function of n, and it is this property that drives most of the mathematics behind all this. Now calculate d, such that
d * e mod [(p - 1) * (q - 1)] = 1

Note that d is called the inverse of e mod (p - 1) * (q - 1), and there is an algorithm called the extended Euclid's algorithm to calculate it. We now have the pieces: the public key is (e, n); the private key is d. Any message must be broken up into chunks that are numeric, and each with a value that is less than n. Each block is then encrypted according to this equation:
C = M^e mod n where M is a message chunk and C is the encrypted chunk

The message can be decrypted by:
M = C^d mod n

The whole thing works by the fact that d cannot be derived from e and n, unless you know the two prime factors of n. Thus e and n can be in a public database; anyone can encipher a message to you, but only you can decipher it. Factoring is difficult; for the system to be robust, p and q should each be at least 100 digits long, and I would currently recommend 200 digits each, as factoring techniques are getting very powerful (a year or so ago a famous 129-digit product of two primes set by MIT as a problem in 1977 was broken by using collaborative computing power across the Internet). In addition, p, q and d should be selected with a number of properties to make the system more secure.

A simple worked example, using very short numbers:

p = 11, q = 5
So n = p * q = 55
Let e = 7
So d * 7 mod (11 - 1) * (5 - 1) = 1
So d * 7 mod 40 = 1
So d = 23
Thus the public key is (7, 55), and the private key 23
The message is 16
To encipher: C = 16^7 mod 55 = 36, and 36 is what is transmitted
To decipher: M = 36^23 mod 55 = 16
It works. Note that if n is 200 digits long, you do need some processing power to make all this actually function. With some very neat algorithms, a PC is more than capable of doing it.