CS 161 Fall 2005 — Computer Security — Joseph/Tygar/Vazirani/Wagner — Notes 18

We will consider the following authentication scheme: the user selects a number $N = P \cdot Q$ product of two large primes, and a number $y = x^2 \bmod N$. The server is given N, y and to login the user must prove that she knows $x : x^2 = y \bmod N$. Notice the similarity between this and the RSA function — here we are squaring instead of cubing to implement our hard to invert function. Indeed, it turns out that computing square roots modulo N is provably as hard as factoring N (as always, this is proved by a reduction. The reduction shows that how to use any algorithm for square root extraction as a subroutine to implement a fast algorithm for factoring).

Before we can state the zero-knowledge protocol and establish its properties, we must state a few facts about numbers which are perfect squares modulo N. Let us restrict our attention to numbers $0 \leq a \leq N-1$ which are relatively prime to N (i.e. $gcd(a,N) = 1$; note that if the gcd is not 1 then it must be P or Q, so such a's are rare and lucky choices that we will not consider). This set of numbers is denoted Z_N^*. For example, for $N = 15$, we would consider the numbers $Z_{15} = \{1,2,4,7,8,11,13,14\}$. Among these numbers only 1 and 4 are perfect squares. Each has four square roots, $\{1,4,11,14\}$ and $\{2,7,8,13\}$ respectively. The square roots come in pairs, e.g. $13 = -2 \bmod 15$ and $8 = -7 \bmod 15$. In fact, for general $N = P \cdot Q$, exactly one quarter of the elements of Z_N^* are perfect squares and every perfect square $a \bmod N$ has four square roots $\overset{+}{-} x$ and $\overset{+}{-} y$. Moreover, multiplying a square by a square gives another square, since $x^2 \cdot z^2 \bmod N = (xz)^2 \bmod N$.

The protocol:

The prover knows $x : x^2 = y \bmod N$. She wishes to prove to the verifier that she knows such a value x.

1. The prover picks a random value $r \bmod N$ and computes $s = r^2 \bmod N$ and sends s to the verifier.

2. The verifier randomly selects one of the following two challenges: I) He asks the prover to send him $\sqrt{s} \bmod N$. II) He asks the prover to send him $\sqrt{sy} \bmod N$.

3. The prover sends either r or $rx \bmod N$ depending upon the challenge.

4. The verifier checks that the received number when squared satisfies the challege.

Let us prove that this protocol provides a zero-knowledge proof of knowledge of a square root of $y \bmod N$. We will show that if the prover does not know a square root of $y \bmod N$ then the honest verifier will catch her cheating with probability at least $1/2$. This will establish that the protocol constitutes a proof of knowledge. And we will show that the verifier cannot extract any extra information from the prover no matter how he deviates from protocol. To do so we will show that for every verifier (no matter how dishonest), there is a simulator that can recreate the verifier's view (his conversation with the prover) without any knowledge of a square root of $y \bmod N$. This will establish that the protocol is zero-knowledge.

Knowledge extractor:

If the prover wishes not to be caught cheating, she must be able to answer both possible challenges of the verifier. We will argue that such a prover must know a square root of $y \bmod N$. For the purposes of the proof let us assume that there is a hypothetical knowledge extractor who can travel backward in time, and after issuing the first challenge and receiving the answer, the knowledge extractor travels back in time and issues the second challenge. By our assumption the prover can answer both challenges and therefore the knowledge extractor receives $u : u^2 = s \bmod N$ and $v : v^2 = sy \bmod N$. Now $w = v/u \bmod N$ is a square root of $y \bmod N$, since $w^2 = v^2/u^2 = sy/s = y \bmod N$. Thus the knowledge extractor can obtain a square root of $y \bmod N$, therefore establishing that the prover must have known a square root of $y \bmod N$. It is important to understand that the knowledge extractor is a hyothetical construct. The protocol requires that the prover only answer if the verifier issues one of the two possible challenges. Also note that a dishonest prover can cheat with probability $1/2$. This probability of cheating can be decreased to $1/2^k$ by repeating the protocol k times.

The Simulator:

What is the verifier's view of the protocol. Note that we are now assuming that the prover is honest and that the verifier is trying to trick the prover into revealing information beyond her knowledge of some square root of $y \bmod N$. Challenge I by the verifier is s a uniformly random perfect square modulo N. Let us show that the second challenge $sy \bmod N$ is also a uniformly random perfect square modulo N. To see this first notice that sy is a perfect square, since it is a product of two perfect squares. Also notice that multiplication by $y \bmod N$ is a permutation of the numbers modulo N. This is because all the numbers we are working with are relatively prime to N, and therefore we can divide by $y \bmod N$ to show that if $sy = s'y \bmod N$ then $s = s' \bmod N$. Thus multiplication by y is a one-to-one mapping from the set of perfect squares to itself, and is therefore a bijection. Thus if we pick s at random among the perfect squares, then $sy \bmod N$ is also uniformly random among the perfect squares modulo N.

The simulator selects a random $r \bmod N$, and with probability $1/2$ sends the verifier $r^2 \bmod N$ and with probability $1/2$ sends the verifier $r^2/y \bmod N$. If the verifier issues challenge I, then in the first case the simulator responds with $r \bmod N$ and otherwise rewinds the simulation and starts again. If the verifier issues challenge II, then in the second case, the simulator responds with $r \bmod N$, and otherwise it rewinds the simulation and starts again. Since the simulator's choice is independent of the choice of challenge issued by the verifier, it follows that the simulation will succeed in satisying the verifier with probability at least $1/2$. The verifier's view is accurately recreated by the simulator, since it just consists of a challenge consisting of a uniformly random perfect square modulo N, followed by a response from the prover consisting of a square root of this number.

Multi-party Protocols:

In a multi-party protocol, we have n players, each with an input, who jointly wish to compute some function of these inputs. For example, in an election protocol, with two candidates — candidate 0 and candidate 1, each player has a single bit which represents his vote, and the protocol must compute the majority of all these n bits. Ideally at the end of the protocol, each player knows only the answer (which is the majority bit) and his own vote, and no further information is leaked during the protocol. We will consider three different models in which such a protocol can be implemented.

Before we do so, let us consider another example of a multi-party protocol: the millionaires problem. Suppose that n millionaires meet at a party and they wish to know who is the wealthiest. However, none of the players wishes to reveal any information about their net worth. Ideally we would like a protocol at the end of which each player learns who is the wealthiest, without learning any further information about the other players. i.e. at the end of the protocol each player knwos her net worth, and the identity of the wealthiest player, and nothing more.

A model in which it is easy to design such multi-party protocols is the trusted party model. For example, an election protocol can be implemented by having each party reveal their vote to the trusted party, who then computes the majority answer and broadcasts the results to all n players. Similarly, in the millionaires problem, each party could reveal their net worth to the trusted party who then figures out which one is the weathiest and broadcasts the answer.

An intermediate model is the honest but curious model. In this model there is no trusted party. However, the players are all honest and will not deviate from the protocol. On the other hand, the players are curious and if any information is revealed during the course of the protocol (without their having to deviate from their prescribed behaviour according to the protocol) then they will try to figure it out. The precise condition stating that the protocol does not leak any information is very much like the zero-knowledge condition. For example, in the voting example, it says that there is a simulator that takes as input player i's vote and the final outcome of the vote, and reconstructs player i's view at the end of the protocol. Protocols for a very wide variety of tasks, including voting and the millionaires problem, can be designed in this model. The design of these protocols require certain tricks that we will not elaborate upon here.

The most general model is one where some of the players are dishonest and can collude with each other to try to compromise the protocol — by either changing the final outcome of the protocol, or by obtaining information beyond what the protocol prescribes each player should have. Using zero-knowledge protocols as a subroutine, any protocol that works in the honest but curious model can be compiled into one that works in the general model. Let us describe the outline of the reduction that establishes this fact. Let us assume that there are n players, and that each has an input x_i. Assume we are given a protocol in the honest but curious model. In this protocol, in the j^{th} step, some player, say player k sends a message based on his input x_k, the messages broadcast during the previous $j-1$ steps, and some random bits flipped by player k. Of these, the messages broadcast are known to the rest of the players, but input x_k and the random bits are private to the k^{th} player. Maintaining this privacy might be essential to the correctness of the protocol in the honest but curious model. In the general protocol, each player k, starts by commiting his input x_k and the string of random bits r_k that he would use during the entire course of the protocol. The commitment could be made by computing $f(x_k), f(r_k)$, where f is a one-way function (this is similar to the commitment made by the user at the beginning of the password identification protocol). Each player still broadcasts his next message to the remaining players. If player k would have broadcast the string m_k in this round, then that is exactly what he would do in the general protocol. The main challenge is convincing the remaining players that m_k is really the message that he would have sent if he were honest. The point is that the true message m_k is a function of x_k, r_k and the previously broadcast messages. In addition to the message m_k, player k also gives a zero-knowledge proof that m_k is the desired function of the x_k, r_k and the previously broadcast messages. This gives a very general theorem showing that there are secure multi-party protocols for a wide array of problems. Needless to say, the resulting protocols are not practical. The value of this result lies in its generality, and the conceptual framework it provides within which one may understand multi-party protocols.