6.876/18.426: Advanced Cryptography May 7, 2003

Lecture 21: Compiling an Honest but Curious Protocol

Scribed by: Jonathan Derryberry

1 Review

In previous lectures, the notion of secure multiparty computing was developed. The setting is that there are m parties, each of which has a private input $x_1, \ldots, x_m$. The goal is to compute $f(x_1, \ldots, x_m, R)$ securely, where R is random coins and "securely" means that no party obtains any more knowledge about other parties' private inputs than could be obtained if all computation were done through a trusted third party. The setting for this problem can be thought of as computers on a network; personal computation is private but all interparty communication is up for grabs.

In previous lectures, Honest but Curious (HBC) security was introduced. In the HBC setting, every party is obliged to follow the protocol, but cannot intentionally "forget" knowledge that it learns during the execution of the protocol. In other words, all parties are curious, in that they try to find out as much as possible about the other inputs despite following the protocol. A protocol is secure in the HBC sense if and only if all parties have no new knowledge at the end of the protocol above what they would have learned from the output of f.

Recall the HBC $\binom{4}{1}$ oblivious transfer (OT) protocol:

$$A \xrightarrow{\quad\quad\quad\quad p \quad\quad\quad\quad} B$$

$$A \xleftarrow{\vec{R};x_1,x_2,p(x_3),x_4|x_i\leftarrow\{0,1\}^n} B$$

$$A \xrightarrow{[\vec{R}\cdot p^{-1}(m_i)]\oplus s_i|i=1,2,3,4} B,$$

where p is a trapdoor permutation, $\vec{R}$ is a random n-bit vector, $\cdot$ denotes the dot product, m_i is either x_i or $p(x_i)$ depending on what B sent in the second step of the protocol, and s_i represents four different secret bits known to A of which B has selected one to learn. In this case, because B applied p to x_3, B learns the value of s_3.

2 Extending HBC

In this lecture, HBC security will be extended to create protocols that are secure even if some subset of the parties are corrupt (as discussed in [GMW87]). In studying the security of these protocols we assume the following

- each party has full knowledge of $A_1, \ldots, A_m$, the algorithms that parties $1, \ldots, m$ are supposed to use during the computation of $f(x_1, \ldots, x_m, R)$
- good guys only have prior knowledge of their own inputs
- no bad guy has prior knowledge of any good guy's input or random coins, which are assumed to come from nature and to be secret
- bad guys may collude as much as they want between each other, including sharing their inputs

Now, to create a protocol that is secure against malicious parties, our strategy has several steps

1. Define HBC solution ($\binom{n}{n}$ private)
2. Produce HBC solution (see last lecture)
3. Produce computationally HBC $\binom{4}{1}$ OT (see the review)
4. Compile HBC protocol into a protocol that is secure against malicious adversaries

To accomplish the last step, we first observe that each party in the protocol only gets to see the messages m that each A_i gets and its answers a to the messages.

3 Enforcing Honest Tape Use in a Malicious Environment

The first difficulty that arises during compilation is that it is hard to determine whether each participant is actually using a random tape, or some cooked up random tape that could potentially allow the extraction of extra knowledge. To guard against this, we could add an opening round of communication in which each A_i broadcasts commitments $C(b_1^{A_i}), C(b_2^{A_i}), C(b_3^{A_i}), \ldots$ to the bits on its random tape. This prevents each A_i from changing its random tape as the protocol progresses because everyone is commited to their random tapes and each player knows how the other is supposed to act, given the contents of the tapes.

At this point, we note that although this prevents the changing of the random tape, it does not ensure that the tape is random to begin with.

3.1 Ensuring the Random Tape's Randomness

To solve this problem, the other parties commit to random guesses about what the contents of the other parties' random tapes are. For example, after A_1 makes commitments $C(b_1^{A_1}), C(b_2^{A_1}), \ldots$ to its random tape, A_2 makes random guesses $C(g_1^{A_2}), C(g_2^{A_2}), \ldots$ regarding the contents of A_1's random tape, and $A_3, \ldots, A_m$ follow suit. Next, all of the guesses for each bit are revealed, and the actual value of A_1's random tape's first bit is declared to be $b_1 \oplus g_2 \oplus \cdots \oplus g_m$. Note that if at least one member of the protocol is honest and makes a random guess then the first bit of the random tape is indeed random. Moreover, note that A_1 has not revealed b_1, so A_1 is the only party to know the value of its random tape at this point if A_1 is honest. Also, A_1 can provide a ZKP that it is behaving in a manner that corresponds to the committed random bits without revealing what they are.

One potential problem with this scheme for generating the contents of the parties' random tapes is that it may be possible for a cheating party to mold its commitment to correlate to the other commitments. This is a problem. For example, if A_m is able to commit to $b_1^{A_1} \oplus g_1^{A_2} \oplus \cdots \oplus g_1^{A_{m-1}}$ then the first bit of A_1's tape could be set to 0 (presumably A_1 would have to collude with A_m for A_m to be able to reveal). However, there are various solutions to this problem. One solution is to simply have the parties reveal their commitments in the reverse order that they broadcasted their commitments. Thus, a correlating cheater would be unable to decommit because it would not know the value of the bit to which it committed. Another solution to the problem is to use mutually independent commitments, which were addressed in a previous lecture, so that such correlation is computationally infeasible.

3.2 Committing to other Tapes

Similarly, everyone can commit to their inputs and work tapes, so that cheating parties cannot decide to change their inputs midway through the protocol or start with a work tape that is not blank. Also, each party sends a commitment to the final state of its worktape. Note that at this point, all of each participant's computation has been committed to. Thus, at the end, if each party A_i provides a ZKP that if the other parties *guessed* the private key to its commitments, then they would be able to *verify* that A_i behaved honestly according to the prespecified algorithms, protocol, and the committed contents of the tapes (assuming starting with the blank tape). Because this is clearly an NP statement, such a ZKP can be given.

3.3 Implementing Communication

When communication is to be broadcast to everyone, there is no problem because there is no concern about who overhears it. However, one useful form of communication in the HBC setting is the ability to "whisper" to other parties so that only one other person hears what you say. This has a natural implementation in the malicious setting. Everyone simply announces their public key at the beginning of the protocol. Thus, if A_1 wants to send A_2 a message m in secret, A_1 broadcasts $E_{A_2}(m)$ to everyone. Note that this has two desirable effects. First, A_2 and only A_2 understands what is said. Second, everyone else receives a commitment to the message, which can be verified as the correct message using a ZKP.

4 Worries

At this point, let us summarize our worries about why this compilation of an HBC protocol might not work:

1. Are different bits of the tape random? *as long as one player is honest and commits to a random bit, the XOR makes the bit random*

2. Parties may be able to correlate commitments to their tapes. *Could reveal backwards, use mutually independent commitments, or require a proof of knowledge of the value committed to*

3. The "agree on a random bit" problem. In a coin-toss-over-the-phone scheme one person knows the value of the random bit before the other, and can abort if they do not like the value.

This last worry has not been addressed directly. One solution to it is to *legislate* that aborting the protocol is not allowed. However, this may pose a problem (what if you are on different planets?).

4.1 Dividing the Secret into Shares

Another solution to this problem is to divide the secret value of the random coin into shares so that any group of less than $\frac{m}{2}$ parties has no information about what the coin is, but any group of more than $\frac{m}{2}$ parties does have enough information to determine what the secret is. Shamir proposed one way of doing this[S79]. The idea is to create a polynomial of the form

$$Q(x) = a_{\frac{m}{2}} x^{\frac{m}{2}} + a_{\frac{m}{2}-1} x^{\frac{m}{2}-1} + \cdots + a_1 x + a_0,$$

where the value of $Q(x)$ is taken modulo a prime $p > m$, and $a_i \in \{0, 1, \ldots, p-1\}$.

Now, let each party $A_1, \ldots, A_m$ have a secret defined as $s_1 = Q(1), \ldots, s_m = Q(m)$. Note that knowing $\frac{m}{2}$ of the secrets yields no knowledge about the remaining values because there is one more degree of freedom in the polynomial. However, note that knowing just one more secret allows the full reconstruction of the polynomial because there are only $\frac{m}{2} + 1$ degrees of freedom for the form of the polynomial.

How can this scheme be used to help us transform an HBC protocol? Each party sends each other party a share of its own private key at the beginning of the protocol[CGMA85]. In other words, party A_i sends $E_{A_1}(Q(1)), \ldots, E_{A_m}(Q(m))$, where Q represents the value of A_i's private key. Note that this allows each party to decrypt exactly one "share" of A_1's private key. Also, note that the correctness of the shares must also be verified in the beginning of the protocol via many zero knowledge proofs (e.g. the NP ZKP: you could guess a private key that corresponds to the public key, and your share would be the share that you received). At the end of the protocol, if some coalition tries to cheat, all of the good guys get together and break the coalition's private keys by sharing the secrets that they received. Note that this method is only effective against coalitions of size $\frac{m}{2}$ or smaller, otherwise the coalition can crack the private keys of all of the honest players.

References

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game - A Completeness Therem for Protocols with Honest Majority. STOC 1987.

[S79] A. Shamir. How to Share a Secret. Communications of the ACM. November, 1979. pp. 612-613.

[CGMA85] Chor, Goldwasser, Micali, and Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of faults. Proceedings of FOCS 85. pp. 383-395.