T-79.511 Special Course on Cryptology / Zero Knowledge: Introduction to Secure Multi-Party Computation

Johan Wallén

November 7, 2001

1 Introduction

Multi-party computation can be seen as a generic framework for solving cryptographic protocol problems. In this framework, we consider m parties and a randomized process, which maps m inputs to m outputs. In this framework, the ith input is thought of as a local input of party i. Similarly, the ith output is thought of as a local output of party i. That is, we are interested in the functionality

$$x \mapsto (f_1(x), \ldots, f_m(x)),$$

where $f\colon (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ is a (possibly randomized) process, f_i denotes the ith component of f, and $x = (x_1, \ldots, x_m)$. Initially, party i holds the input x_i, and in the end, it receive the output $f_i(x_1, \ldots, x_m)$.

In the presence of a *trusted third party*, this problem can easily be solved. First, each party picks its local input x_i and sends it to the trusted party using a secure channel. Then the trusted party computes the functionality, and finally sends the ith output to the party i. Even in this ideal model, a coalition of parties can jointly agree on which inputs to use, share all the outputs received by members of the coalition (and perform arbitrary computation based on these outputs), or abort the protocol (by refusing to participate in it). A protocol for *secure multi-party computation* lets the parties *simulate* the trusted third party of the *ideal model* in a *real model*, where parties can deviate from the protocol in any computationally feasible way.

Note that the abortion of the protocol of the ideal model becomes more involved when translated to a real model. In a real model, parties might be able to abort the protocol at a later stage, say, when some of the parties have received their outputs. In general, this is difficult to prevent. It, however, turns out that early abortion can be *effectively prevented*, if a majority of the parties are honest. Thus, we will consider two real models. The first does not consider early abortion as a break of security, but is instead secure if *at least one* of the parties is honest. In the second model, we do consider early abortion as a break of security, but we instead assume that *more than half* of the parties are honest. Note that only the first real model is interesting for two party protocols.

We will proceed as follows. First, we will define one *semi-honest* and two *malicious* models including their *security requirements*. The security requirements are specified using *ideal models*, and security is specified using *simulation arguments* showing that any feasible adversary in a real model can be simulated in the corresponding ideal model. Then we will discuss the underlying complexity assumptions and some tools. After that, we will proceed to constructions secure in the semi-honest model. Finally, we discuss how a protocol secure in the semi-honest model can be transformed to protocols secure in the two malicious models by forcing semi-honest behavior. This is done using two *compilers*. The first compiler transforms any protocol secure in the semi-honest model to a protocol secure in the first malicious model (the one allowing abort). The second compiler transforms secure protocols in the first malicious model to secure protocol in the second malicious model (the one with a honest majority).

Our discussion follows the one in [5]. Due to space constraints, we do not prove the correctness of the constructions. We refer to [5] or the original papers for proofs. Throughout this survey, we assume some familiarity with basic zero knowledge. We refer to the other surveys in this seminar for this background.

2 Models

2.1 Notation

An *m-ary functionality* $f\colon (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ is a randomized process mapping m sequences of inputs to m sequences of outputs. Initially, party i holds the input x_i, and wishes to obtain the ith output denoted $f_i(x_1,\dots,x_m)$. For simplicity, we make the following assumptions.

- The inputs have the same length: $|x_i| = |x_j|$.
- The functionality is computable in polynomial time (with respect to $|x_i|$).
- Security is measured in terms of the length of the inputs.
- The functionality is deterministic.

The last assumption is not a loss of generality: We can always consider a deterministic function getting an explicit random string as an additional input, and augment the protocol with a coin-tossing stage. For convenience, we also fix the number of parties to $m > 1$.

Let $[m] = \{1,\dots,m\}$ denote the set of all parties. For any $I = \{i_1,\dots,i_k\} \subset [m]$ let $f_I(x)$ denote the sequence $(f_{i_1}(x),\dots,f_{i_k}(x))$. Similarly, for $x = (x_1,\dots,x_m)$, we let $x_I = (x_{i_1},\dots,x_{i_k})$. We also define $\bar{I} = [m] - I$.

We say that an m-party protocol Π *computes* f, if the output of Π is the same as that of f.

Let $X = \{X_n\}$ and $Y = \{Y_n\}$ be probability ensembles. We say that the ensembles are *computationally indistinguishable*, written as $X \overset{c}{\sim} Y$, if no efficient algorithm can tell them apart with significant probability.

To simplify notation, we usually let the malicious parties be the first ones. That is, the coalition of malicious players is $I = \{1,\dots,t\}$ for some t.

It is convenient to think of a single adversary controlling the coalition.

2.2 The semi-honest model

In the semi-honest model, the parties follow the protocol, but might try to get additional information from all received messages. Intuitively, a protocol is secure in the semi-honest model, if it *privately* computes the functionality—that is, no coalition gain any additional information from the protocol run.

Definition 1 (The semi-honest model). Let f be an m-ary functionality and let Π be a protocol for f. The *view* view_i^Π of the ith party during an execution of Π on input $x = (x_1,\dots,x_m)$ is defined by $\mathsf{view}_i^\Pi(x) = (x, r, m_1,\dots,m_t)$, where r represents the outcome of the ith party's internal coin tosses, and m_j is the jth message received. Similarly, the view of the coalition $I = \{i_1,\dots,i_k\} \subset [m]$ is defined by $\mathsf{view}_I^\Pi(x) = (I, \mathsf{view}_{i_1}^\Pi(x),\dots,\mathsf{view}_{i_k}^\Pi(x))$. We say that the protocol Π *privately computes* f (in the semi-honest model), if there is a polynomial-time algorithm S such that for every $I \subset [m]$,

$$\{S(I, x_{i_1},\dots,x_{i_t}, f_I(x))\}_x \overset{c}{\sim} \{\mathsf{view}_I^\Pi(x)\}_x.$$

2.3 The first malicious model

In the first malicious model, *early abortion* is not considered to be a break of security, but we instead let an arbitrary number of parties deviate from the protocol. In this model, malicious parties may refuse to enter the protocol, deviate from the protocol in any computationally feasible way, and may abort the protocol prematurely.

We first define an ideal model reflecting the security requirements.

Definition 2 (The ideal model). Let f be an m-ary functionality, $I = \{i_1,\dots,i_t\} \subset [m]$ be a coalition, and let C be a deterministic polynomial-size circuit family. We let (I, C) represent an adversary. The *joint execution under* (I, C) on input $x = (x_1,\dots,x_m)$, $\mathsf{ideal}^1_{f,(I,C)}(x)$, is defined as follows. If $C(x_I) = \perp$, the coalition aborts immediately and the output is

$$(C(x_I, \perp), \perp, \dots, \perp).$$

Otherwise, if $C(x_I, f_I(C(x_I), x_{\bar{I}})) = \perp$, where $\bar{I} = [m] - I$, the coalition aborts the protocol after it has received its output, but before the other parties have received their input. In this case, the output is

$$(C(x_I, f_I(C(x_I), x_{\bar{I}}), \perp), \perp, \dots, \perp).$$

Otherwise, the output is

$$(C(x_I, f_I(C(x_I), x_{\bar{I}})), f_{\bar{I}}(C(x_I), x_{\bar{I}})).$$

Note that we only consider deterministic adversaries. This is no real loss of generality. Intuitively, we could find the best coin sequence of a randomized adversary and then fix the sequence to that value.

The definition of a real adversary is much simpler, namely:

Definition 3 (The real model). Let f and I be as above, Π be a protocol for f, and C be a deterministic polynomial-size circuit family. The *joint execution under* (I, C) on input x, denoted $\mathsf{real}_{\Pi,(I,C)}(x)$, is the output sequence resulting from the interaction, when the messages of the parties in I are computed according to C, and the messages of parties not in I are computed according to Π.

We are now ready to define security in the first malicious model.

Definition 4 (Security). A protocol Π securely computes f in the first malicious model, if there is a polynomial-time transformation of polynomial-time circuit families $A = \{A_n\}$ for the real model (Definition 3) into polynomial-size circuit families $B = \{B_n\}$ for the ideal model (Definition 2) such that for every $I \subset [m]$,

$$\{\mathsf{ideal}^1_{f,(I,B_n)}(x)\}_{n,x} \overset{c}{\sim} \{\mathsf{real}_{\Pi,(I,A_n)}(x)\}_{n,x},$$

where x range over $(\{0,1\}^n)^m$.

That is, any feasible behavior of an adversary in the real model can be simulated in the ideal model.

2.4 The second malicious model

In the second model, we do consider abortion to be a break of security. On the other hand, we only consider coalitions of size less than $m/2$. Thus, we only consider models with a *honest majority.*

Definition 5 (The ideal model). Let f be an m-ary functionality, $I = \{i_1, \ldots, i_k\} \subset [m]$, and C a polynomial-size circuit family. We say that (I, C) is *admissible*, if $|I| < m/2$. The *joint execution* $\mathsf{ideal}^2_{f,(I,C)}(x)$ *under* (I, C) in the second ideal model on input x, is defined by

$$(C(x_I, f_I(C(x_I), x_{\bar{I}})), f_{\bar{I}}(C(x_I), x_{\bar{I}})).$$

The real model is defined exactly as in the first model, but here we only consider *admissible* adversaries. The security for the second model is defined exactly as in the first, except that we only consider coalitions I with $|I| < m/2$, and replace the first ideal model with the second ideal model.

3 Tools

In this section, we briefly and very informally discuss some underlying assumptions and tools. We refer to textbooks such as [4] for a more thorough discussion.

3.1 Complexity assumptions

Throughout this survey, we assume the existence of *trapdoor permutations.* These are collections of one-way permutations with the additional property that a function is easy to invert given a *trapdoor* as additional input. Note that trapdoor permutations in particular implies the existence of one-way functions.

3.2 Commitment schemes

A *commitment scheme* is a uniform family of probabilistic polynomial time circuits $\{C_n\}$ such that the following two conditions hold.

- **Perfect unambiguity:** For every n, the supports of $C_n(0)$ and $C_n(1)$ are disjoint.
- **Computational secrecy:** The ensembles $\{C_n(0)\}_n$ and $\{C_n(1)\}_n$ are computationally indistinguishable.

In the sequel, we will let $C_n(b, r)$ denote the output of C_n on the input bit b and using the random string r. Note that $C_n(0, r) \neq C_n(1, s)$ for all r, s.

Commitment schemes can be based on any one-way function.

3.3 Oblivious transfer

Let k be a fixed integer (we will use $k = 4$). In the *oblivious transfer* functionality OT^k_1 [8, 3], the inputs of party 1 (the sender) are k bits b_i, $i = 1, \ldots, k$, and the input of party 2 (the receiver) is an integer $i \in \{1, \ldots, k\}$. The OT_1^k functionality is defined by

$$\mathrm{OT}_1^k((b_1, \ldots, b_k), i) = (\epsilon, b_i),$$

where ϵ denotes the empty string. The functionality OT_1^k can be privately implemented (in the semi-honest model) using any family of trapdoor permutations.

4 Constructions for the semi-honest model

4.1 Reduction to the two party case

Without loss of generality, we assume that the functionality f is given as a Boolean circuit consisting of AND and XOR gates with fan-in 2. This circuit can be considered as an arithmetic circuit over GF(2). The m parties will run a circuit evaluation protocol to obtain their parts of the output. The core of this protocol consists of the emulation of the addition and multiplication gates. When simulating a gate, each party will hold a *share* of each input value so that all the shares sum up to the correct input value. After the simulation of the gate, each party will hold a similar share of the correct output value.

Proceeding trough an addition gate is trivial. If the input shares of party i are (a_i, b_i), it just sets it output share to $c_i = a_i + b_i$. This produces the correct value, since $\sum_i a_i + \sum_i b_i = \sum_i c_i = a + b$. In addition, the c_is are uniformly distributed, provided the a_is and b_i where.

Thus, the main problem is the emulation of multiplication gates. Each party holds uniformly distributed shares a_i, b_i, such that $a = \sum_i a_i$ and $b = \sum_i b_i$. The subject of the gate simulation is to produce uniformly distributed shares c_i such that $c = \sum_i c_i = ab$.

Observe that $\left(\sum_{i=1}^m a_i\right)\left(\sum_{i=1}^m b_i\right) = \sum_i a_i b_i + \sum_{i<j}(a_i b_j + a_j b_i) = (1 - (m-1))\sum_i a_i b_i + \sum_{i<j}(a_i + a_j)(b_i + b_j) = m\sum_i a_i b_i + \sum_{i<j}(a_i + a_j)(b_i + b_j)$. In this expression, each party can compute by itself the term $m a_i b_i$, whereas the terms $(a_i + a_j)(b_i + b_j)$ for $i < j$ can be computed by invoking a *two-party* version of the multiplication emulating protocol. To be precise, each pair of parties $\{i, j\}$ invoke a two-party version of the gate emulation protocol with input shares (a_i, b_i) and (a_j, b_j). As a result of this invocation, the party i obtains the shares $c_i^{\{i,j\}}$ and party j obtains $c_j^{\{i,j\}}$. Finally, party i sets its output share to $c_i = m a_i b_i + \sum_{j \neq i} c_i^{\{i,j\}}$.

4.2 The two-party case

We will reduce the problem of privately computing $c_1 + c_2 = (a_1 + a_2)(b_1 + b_2)$ to oblivious transfer OT_1^4. The construction is as follows.

Initially, party i holds $a_i, b_i \in \{0, 1\}$ for $i = 1, 2$. Party 1 uniformly selects $c_1 \in \{0, 1\}$. Then the two parties invoke a protocol for OT_1^4. Party 1 plays the role of the sender with input $(c_1 + a_1 b_1, c_1 + a_1(b_1 + 1), c_1 + (a_1 + 1)b_1, c_1 + (a_1 + 1)(b_1 + 1))$. Party 2 plays the role of the receiver with input $1 + 2a_2 + b_2 \in \{1, 2, 3, 4\}$. The output of party 1 is c_1, whereas the output of party 2 is the bit received from the OT_1^4 protocol.

4.3 Putting it all together

In the full circuit emulation protocol, party i initially holds the input $x_i^1 \cdots x_i^n$. The protocol proceeds as follows.

- **Sharing the inputs:** Each party i splits and shares its input bits. For $j = 1, \ldots, n$ and $k \neq i$, party i selects a random bit $r_k^{(i-1)n+j}$ and sends it to party k as its share of the input wire $(i-1)n + j$. Party i sets its own share of the $(i-1)n+j$th input wire to $x_i^j + \sum_{k \neq i} r_k^{(i-1)n+j}$.
- **Circuit emulation:** The parties emulate the individual gates one by one as described above.
- **Recovering the outputs:** For each output wire of the circuit, each party sends its share of the wire to the associated party. Each party recovers its output bits by adding together its own share and all shares received in this step.

This rough outline of the constructions can be maid formal and shown correct leading to the following result.

Theorem 6. Suppose that trapdoor permutations exists. Then every m-ary functionality is privately computable (in the semi-honest model).

5 Forcing semi-honest behavior

5.1 Broadcast

In the semi-honest model, we (implicitly) used a point-to-point communication model with secure channels. To simplify the discussion, we will describe the two compilers in a communication model consisting of a single broadcast channel. Thus, the full sequence of transformations for transforming a protocol Π_0 secure in the semi-honest model to protocols Π_1 and Π_2 secure in the two malicious models is as follows.

- **Precompilation:** The protocol Π_0 privately computing f in the point-to-point model is compiled into a protocol Π_0' privately computing f in the broadcast model (that does not contain secure channels).

- **Compiling to the first model:** Using the first compiler, the protocol Π_0' is compiled to the protocol Π_1' secure in the first malicious model (using broadcast).

- **Compiling to the second model:** Using the second compiler, Π_1' is compiled to the protocol Π_2' secure in the second malicious model (using broadcast).

- **Postcompilation:** Finally, the protocols Π_1' and Π_2' are transformed to secure protocols Π and Π_2 for the standard point-to-point model.

Note that the first step is easily implemented using standard techniques. The last step can be implemented using *authenticated Byzantine agreement.*

Definition 7. Assume synchronous point-to-point communication and a signature scheme infrastructure. Party 1 has an input bit σ. The objective of *authenticated Byzantine agreement* is to let all honest parties agree on the value of this bit. If party 1 is honest, they must agree on the actual value of σ. If party 1 is not honest, they may agree on any value (as long as they agree).

Authenticated Byzantine agreement can be implemented using the following construction.

- Phase 0: Party 1 signs σ and sends it to all parties.

- Phase $i = 1, \ldots, m$: Each honest party inspect the messages received during phase $i-1$. A message is *admissible* if it has the form $(v, s_{p_0}, \ldots, s_{p_{i-1}})$, where $p_0 = 1$, all the p_is are distinct, and for $j = 0, \ldots, i-1$, s_{p_j} is a valid signature to $(v, s_{p_0}, \ldots, s_{p_j})$ of party p_j. Such an admissible message is said to be an *authentic* $(v, i-1)$-message, or simply v-message.

  If the party finds an authentic $(v, i-1)$-message, it signs this message, appends the signature to it, and sends the resulting message to all parties. Note that this results in an authentic (v, i) message.

- Each party (except 1) proceeds as follows. If it has received both an authentic 0-message and an authentic 1-message, it decides that party 1 is malicious and outputs a default value (say 0). Otherwise, if it has received an authentic v-message, it outputs the value v. Otherwise, is decides that party 1 is malicious and outputs a default value (say, 0).

5.2 The first compiler

There are basically three ways a malicious party can deviate from the protocol: He can replace his input or random string at a later stage of the protocol, use a random string that is not uniformly distributed, or deviate from the circuit evaluation protocol. To force semi-honest behavior, we will augment the protocol with an input commitment phase, a coin generation phase, and an authenticated computation protocol.

5.2.1 Input commitment

During the input commitment phase, each party will commit to its input $x_i = x_i^1 \cdots x_i^n$. For each input bit x_i^j, party i picks an commitment string r and broadcasts the commitment $C_n(x_i^j, r)$. Thus, the malicious parties can jointly agree on their inputs, or decide not to enter the protocol at all. After the commitment phase, the parties cannot alter their inputs.

5.2.2 Coin generation

During the coin generation phase, random coins for the simulation of the semi-honest protocol will be generated for each party. Specifically, we are interested in the randomized functionality

$$(1^n)^m \mapsto ((\sigma, r), C_n(\sigma, r), \ldots, C_n(\sigma, r)).$$

That is, one of the parties obtains a uniformly distributed bit σ and an commitment string r. The other parties obtains commitments to σ. This functionality can be implemented in the first malicious model (with broadcast) by combining standard coin tossing protocols [1] and commitment schemes. The resulting so called *coin tossing into the well* protocol can be used to generate random coins for each party, and commitments for these coins. After this stage, all parties have uniformly distributed and fixed coins.

5.2.3 Authenticated computation

In the authenticated computation functionality, one party got the input α and the random string r. All parties share a common input β. In addition, we have two polynomial-time computable functions f and h. The function h captures information about α known by all parties, whereas f captures the desired computation. We are interested in the functionality $((\alpha, r, \beta), (h(\alpha, r), \beta), \ldots, (h(\alpha, r), \beta)) \mapsto (\epsilon, f(\alpha, \beta), \ldots, f(\alpha, \beta))$. For simplicity, we can assume that h is one-to-one with respect to its first argument: for all $\alpha \neq \alpha'$ and all r, r', $h(\alpha, r) \neq h(\alpha', r')$. This functionality can be securely computed in the first malicious (broadcast) model:

Party 1 gets input (α, r, β) and the other parties gets input (u, β), where $u = h(\alpha, r)$. First, party 1 broadcasts $v = f(\alpha, \beta)$. Then, the parties invoke $m - 1$ instances of a zero knowledge proof system, where party 1 proves to each party separately that there are x, y such that $u = h(x, y)$ and $v = f(x, \beta)$. If a verifier rejects a proof, it aborts the protocol.

The parties will use the authenticated computation protocol to simulate one step of the underlying protocol Π_0'. The party j, which is supposed to send a message in the underlying protocol, plays the role of party 1. The inputs α, β, r, and the functions f, and h are set as follows.

- α is set to the concatenation of the party's original input x^j and random string r^j (in the underlying protocol). r is set to the concatenation of the randomizations used in the commitments. $h(\alpha, r)$ is the concatenation of all the corresponding commitments.
- β is set to the concatenation of all messages sent over the broadcast channel so far.
- The function f is the computation which determines the next message to be sent in the underlying protocol.

The result of this emulation is that each party can verify that the message sent indeed was computed according to the underlying protocol.

5.3 The second compiler

The idea behind the second compiler is to augment the protocol secure in the first malicious model with a *verifiable secret sharing* scheme. If a party aborts the protocol after the sharing step, the honest majority can recover the input of the aborting parties and complete the protocol. This effectively prevents early abortion. Malicious parties can, of course, abort during the verifiable secret sharing step, but this is essentially equivalent to not entering the whole protocol at all.

Let $t \leq m$ be positive integers. Recall that a t-out-of-m secret sharing scheme consists of two algorithms, $G_{m,t}$ and $R_{m,t}$. Here, $G_{m,t}$ is the share generation algorithm mapping a single bit to m sequences of shares. $R_{m,t}$ is the recovering algorithm mapping sequences of pairs in $[m] \times \{0,1\}^*$ to a single bit. The algorithm must satisfy the following two conditions.

- For any bit σ and sequence $(s_1, \ldots, s_m)$ in the range of $G_{m,t}(\sigma)$ and any $\{i_1, \ldots, i_t\} \subseteq [m]$, $R_{m,t}((i_1, s_{i_1}), \ldots, (i_t, s_{i_t})) = \sigma$.
- For any $I = \{i_1, \ldots, i_{t-1}\} \subset [m]$, the I components of $G_{m,t}$ are independent of σ.

Definition 8 (VSS [2]). Let $G_{m,t}$ be a share generation algorithm producing shares of length ℓ. Let $\{C_n\}$ be a bit commitment scheme, and define $C_n(\sigma_1 \ldots \sigma_\ell, r_1 \cdots r_\ell) = C_n(\sigma_1, r_1) \cdots C_n(\sigma_\ell, r_\ell)$. Consider the functionality $((\sigma, 1^n), 1^n, \ldots, 1^n) \mapsto ((s, \rho), (s_2, \rho_2, c), \ldots, (s_m, \rho_m, c))$, where s consists of all the shares, $s = (s_1, \ldots, s_m) = G_{m,t}(\sigma)$, $\rho = (\rho_1, \ldots, \rho_m)$ is uniformly distributed, and

$c = C_n(s_1 \cdots s_m, \rho_1 \cdots \rho_m)$ consists of all the commitments. Any m-party protocol for securely computes this functionality in the first malicious model is called a *verifiable secret sharing scheme*.

In the compiler, we will use the parameters $(m, \lceil m/2 \rceil)$.

Note that verifiable secret sharing schemes do exist, since any functionality can be securely implemented in the first malicious model.

During the sharing phase, each party will share its input with all the other parties. Note that each party can prove the validity of its share simply by revealing ρ_i. If any party aborts, the honest majority can recover its input and complete the protocol. Since more than half of the parties are honest, a malicious minority cannot recover any input. On the other hand, the honest majority (who can recover the secret) will not recover the input of any non-aborting party, since the majority is assumed to follow the protocol.

6 Conclusions

By putting all the constructions together, we get the following very general plausibility result.

Theorem 9 ([9, 7, 6]). Suppose trapdoor permutations exist. Then every m-ary functionality can be securely computed in both malicious models.

References

[1] M. Blum. Coin flipping by phone. In *IEEE Spring COMPCOM*, pages 133–137, 1982.

[2] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th IEEE Symposium on Foundations of Computer Science*, pages 383–395, 1985.

[3] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *CACM*, 28(6):637–647, 1985.

[4] O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2001.

[5] O. Goldreich. Secure multi-party computation. Manuscript, Version 1.3, 24 June 2001.

[6] O. Goldreich, S. Micali, and A. Widgerson. How to play any mental game—a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.

[7] O. Goldreich, S. Micali, and A. Widgerson. Proofs that yield nothing but their validity or all languages in NP have zero knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[8] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Labroratory, 1981.

[9] A. Yao. How to generate and exchange secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.