Foundations of Cryptography

Notes of lecture No. 14 & 15

Notes taken by Ran Canetti and Benny Pinkas

Summary

In this lecture we discuss the problem of constructing a protocol for computing a function whose input is divided among a number of parties. This protocol must retain the privacy of data of each of the participants, even when some of the other parties try to get this data. We define the notion of a protocol that *privately computes* a function. We show a protocol that *privately computes* any function, according to the given definition, when more than half of the parties are *honest*. The time complexity of the protocol is polynomial in the complexity of the function.

1 Introduction

The general goal of distributed computing is to develop protocols for (distributively) computing functions whose input is scattered among the processors. If all processors follow their predetermined programs then the existence of such protocols follows immediately from the specification of the corresponding function, and the only challenge is in improving the (message and time) complexities of these protocols. However, the situation is much more complex if some of the parties may deviate from their predetermined program in certain ways, either because of a fault or trying to get some secret information from the other parties. A natural model of such misbehavior allows *faulty* parties to deviate from their predetermined program in an arbitrary, but (probabilistic) polynomial-time manner. When such faults are present, it is no longer clear if there exist protocols that are *correct* in the sense that they terminate with each of the non-faulty parties having the value of the function. Furthermore, it is not clear whether correct protocols can offer the maximum possible *privacy* of local inputs allowed by the function. (Namely, whether it is possible to restrict what the faulty parties can learn from an execution of the protocol to the value of the function. In particular, the faulty parties cannot learn about the input of the non-faulty parties more than the function value.)

The main result presented here is an affirmative resolution of the above problem. If secure trapdoor encryption functions exist and as long as more than half of the parties remain non-faulty, *every* function has a *correct* fault-tolerant protocol that offers the maximum degree of *privacy*. The complexity of the protocol is polynomial in the time-complexity of the function. Furthermore, there exists a polynomial-time algorithm that on input a Turing-Machine specification of the function outputs such a protocol.

A protocol problem is a specification of the input and expected output (function value) of each party. Following is an example to such a problem:

Elections (in the participation of n parties, denoted $P_1, P_2, \cdots, P_n$).

Local **input** for each P_i: A vote v_i.

Local **output** for each P_i: $v_{\max}$, which is the vote that had the maximal support (appeared more than any other vote in the sequence $v_1 ... v_n$.)

A solution for a protocol problem is a protocol which satisfies the following conditions:

1. *Correctness:* Even in the presence of faulty parties, all the non-faulty parties get an identical output which corresponds to the inputs for which the parties committed themselves, or else they receive a message that the protocol was interrupted.
2. *Privacy:* Even for a collaboration of all the faulty parties, whatever they can efficiently compute after participating in the protocol, they can also efficiently compute from their local inputs and local outputs (function value).

In order to illustrate the meaning of the above requirements, consider an environment in which in addition to the participating parties there is a *trusted party*. In the ''trusted party environment'', each party sends its input to the trusted party using a *secure* channel; then the trusted party computes the output and sends it to all the parties. Since the trusted party is non-faulty, the computed value of the function is the correct one. It is also clear that no collaboration of faulty parties can learn more than what can be computed from the final output and their local inputs. A solution to a protocol problem should have the same effect as a computation in the ''trusted party environment''. We will see that such a solution can be achieved even if none of the parties is trusted by all the others.

2 The Formal Setting

There are n parties, $P_1, P_2, ..., P_n$. Each party P_i has a local input x_i, and all other parties have a commitment $c(x_i)$ of P_i to the value of x_i. At the end of the protocol each party should have an output $f_i(x_1, ..., x_n)$.

Remarks

- The requirement for a commitment on the input values is important in order to commit each party to a *single* input during the execution of the protocol. This greatly simplifies the definition since without it, it is not clear to which input values (of the faulty parties) should the computed function value correspond. Additionally, there are situations where this is a natural requirement, and in any case such a commitment can be enforced by a preliminary protocol.
- Without loss of generality we can assume that all parties compute the same output function f. We can create such a function from the local output functions f_1 through f_n, by letting

$f(x_1, \cdots, x_n) \triangleq E_1(f_1(x_1, \cdots, x_n)) \circ \cdots \circ E_n(f_n(x_1, \cdots, x_n))$ where E_i is a secure public-key encryption for which only P_i has the decryption key.

- For simplicity of exposition, we will consider the domain of f as being the n-fold Cartesian product of $\{0,1\}^n$, namely $(\{0,1\}^n)^n$. That is, each of the n parties has an input n bits long. Therefore the notion of ''polynomial'' is in regard only to a single parameter, n. The results presented hold also if this is not the case. That is, the protocol is polynomial in regard both to the number of parties and the length of each input .
- Furthermore, without loss of generality, in the rest of the paper we consider only binary functions (functions into $\{0,1\}$).
- We will assume that f is polynomially computable. If this is not the case then we can use cryptographic systems whose security parameter is the complexity of computing f (e.g. if f is exponential, then the keys of the cryptographic system will have a length exponential in the length of the input). In any case, the complexity of the protocol will be polynomial in the complexity of the function, and so is the complexity of the adversary.
- We assume the existence of a *broadcast channel*, and that all the processors are synchronized. If either feature is not available, it can be simulated using Byzantine Agreement methods, as long as less than a third of the parties are faulty.

An *honest* party is a party which runs the original program. A *faulty* party is one which may deviate from the original protocol in an arbitrary but (probabilistic) polynomial-time manner. Faulty protocols may also collaborate with each other. The results presented here hold if the faulty parties are assigned *before* the execution of the protocol starts.

Notations:

- $ex_S(A,B,(x_1, \cdots, x_n)) \triangleq (\alpha_1, \cdots, \alpha_n)$,

  where A,B are local programs, $S \subseteq \{1,...,n\}$, and α_i is the output of P_i in an execution where for $j \in S$ processor P_j executes protocol A (with local input x_j), and for $j \notin S$ processor P_j executes protocol B (with local input x_j). Note that when A or B are probabilistic , ex_S is a random variable.
- $(\alpha_1, \cdots, \alpha_n)_i \triangleq \alpha_i$
- $(\alpha_1, \cdots, \alpha_n)_S \triangleq (\alpha_{i_1}, \cdots, \alpha_{i_s}), \quad i_j \in S$

We now define the notion of correct fault-tolerant protocols that retain the maximum degree of privacy given by a function. A method for constructing such protocols is introduced later on, and is the main result presented in this lecture.

Definition1: Let $f:(\{0,1\}^n)^n \rightarrow \{0,1\}$ be a polynomially computable function. A protocol π *privately computes f in the presence of t faulty parties* if the following conditions hold:

1. $ex_{\{1,...,n\}}(\pi,\pi,(x_1,\cdots,x_n))=(f(x_1,\cdots,x_n),\cdots,f(x_1,\cdots,x_n))$.

   (Namely in the absence of faulty parties the output of the protocol is the correct function value.)

2. *Correctness:* For every set S of $n-t$ honest parties:

   2.1 For every efficient (polynomial and non-uniform) algorithm A (of the faulty parties)

   $$ex_S(\pi,A,(x_1,\cdots,x_n))_S = \begin{cases} f(x_1,\cdots,x_n),\cdots,f(x_1,\cdots,x_n) \\ \perp,\cdots,\perp \end{cases}$$

   (That is, all honest parties have the same output which is either the correct function value or a special symbol ($\perp$), denoting that the execution of the protocol was interrupted.)

   2.2 For every such algorithm A, there is an efficient algorithm M, such that if

   $ex_S(\pi,A,(x_1,\cdots,x_n))_S=\perp,\cdots,\perp$ then $M((x_1,\cdots,x_n)_{\bar{S}})\ (ex_S(\pi,A,(x_1,\cdots,x_n)))_{\bar{S}}$, where $\bar{S}$ denotes $\{1,...,n\}\backslash S$, and

   $\equiv$ means ‘‘polynomially indistinguishable’’.

   (Namely, when the protocol is prematurely terminated, whatever the faulty parties can efficiently compute at this stage can also be efficiently computed from their local inputs; the faulty parties did not learn anything about the inputs of the honest parties. Therefore the termination of the protocol does not depend on the input of the honest parties!)

3 *Privacy:* For every set of $n-t$ honest parties S, and for every efficient algorithm A, there is an efficient algorithm M, such that if $ex_S(\pi,A,(x_1,\cdots,x_n))_S=f(x_1,\cdots,x_n),\cdots,f(x_1,\cdots,x_n)$ then $M((x_1,\cdots,x_n)_{\bar{S}},f(x_1,\cdots,x_n))\equiv(ex_S(\pi,A,(x_1,\cdots,x_n)))_{\bar{S}}$.

(That is, whatever the faulty parties can efficiently compute after the protocol has successfully terminated, can be efficiently computed from their local inputs and the function value.)

Parts 2.2 and 3 can be combined by stating that $M((x_1,\cdots,x_n)_{\bar{S}},ex_S(\pi,A,(x_1,\cdots,x_n)))_S)\equiv(ex_S(\pi,A,(x_1,\cdots,x_n)))_{\bar{S}}$. That is, whatever the faulty parties can efficiently compute after any execution of the protocol, can be efficiently computed from their local inputs and the their local outputs (at this run).

For the sequel we also need the notion of *semi-honest* parties.

Definition2: A *Semi-honest* party is a party that

- Uses for its random tape the output of independent unbiased coin tosses.
- In each communication round sends exactly the same message as instructed in the protocol. (The protocol defines a unique message to be sent, as a function of the initial input, the random tape, and the communication received so far. The work tape need not be considered since it is also uniquely defined by those three parameters.)

- Does not listen to any communication other than those sent to him.

In other words, semi-honest parties may deviate in their *internal* computation from the protocol, but the messages they send are in accordance with the protocol. A semi-honest party may also be regarded as one that executes the original protocol but tries to compute as much additional information as possible; this may be the case in some realistic situations and therefore is an important notion by itself. Note that there is no way for an outside observer to distinguish between an honest and a semi-honest party.

Although the last requirement may seem unnatural, it can be easily enforced using a public-key encryption system: whenever a party wants to send message m to party P_i, it will send $E_i(m)$, thus enabling only P_i to read it (section 5 has the details). An alternative definition of a *semi-honest* party does not include the last requirement, allowing the *semi-honest* party to listen to conversations of other parties; in this case the above use of a public-key encryption system would have to appear in almost [1] any protocol for *semi-honest* parties.

Semi-honest parties will be used in the following construction as an intermediate stage between honest and faulty parties.

3 The Main Result

Theorem 1. Let $f:(\{0,1\}^n)^n \to \{0,1\}$ be a polynomially computable function, and let $t < \frac{n}{2}$. If there exists a one-way permutation with a trapdoor then there exists an efficient protocol for n parties, $P_1 ... P_n$, that *privately computes* f in the presence of t faulty parties.

Proof. The theorem is proved using two propositions, which are also interesting by themselves. In proposition 1 (see section 4), we show a protocol that *privately computes* f according to the definition, if all the parties are honest or semi-honest (note that there is no limit on the number of semi-honest parties). In proposition 2 (see section 5) we show a method of ''compiling'' such a protocol, π, into another protocol, $C(\pi)$, that *privately computes* f in the presence of $t < \frac{n}{2}$ faulty parties. The theorem follows. ■

4 A Protocol For Semi-honest Parties

Proposition 1. Let $f:(\{0,1\}^n)^n \to \{0,1\}$ be a polynomially computable function. Then there exists an efficient protocol for n parties, $P_1 ... P_n$, that *privately computes* f if all the parties are honest or semi-honest.

Proof outline. We will first show a protocol that *privately computes* f using as a ''subroutine'' a two-party protocol for a simpler problem. We then implement this two-party protocol using the Oblivious Transfer (OT) scheme, and finally we show an OT protocol, assuming the existence of one-way permutations with a

[1] It would be unnecessary to use this ''precaution'' if the messages sent do not reveal any additional information to what can be efficiently computed from the input and output of any coalition of parties. For instance, if $f(x_1,...,x_n) = x_1 \circ ... \circ x_n$.

trapdoor.

4.1 The general protocol

Since f is polynomially computable there exists a boolean circuit of polynomial-size that computes f. The input to f is of size n^2 (n bits to each party), thus the circuit is polynomial also in n. Assume, without loss of generality, that the circuit consists only of $\neg$ and $\wedge$ gates ($a \vee b = \neg(\neg a \wedge \neg b)$) and that the maximum fan-in of a gate is two. We consider these operators as in Z_2 algebra: the $\neg$ operator is replaced by +1, and the $\wedge$ by a Z_2 multiplication. Each party has this circuit, and his n private input bits (out of the n^2 input bits to the circuit). Consider a partial order of the gates in the circuit, so that a gate g_1 precedes gate g_2 if an input line of g_2 collides with an output line of g_1. Let $g_{\max}$ be the gate whose output value is the value of f. We show a protocol that, assuming the existence of a specific two-party protocol (described in section 4.2), allows each party to compute the output of the circuit without ''revealing its private input''. That is, for each coalition C of a subset of the parties, whatever C can efficiently compute after the termination of the protocol, can be efficiently computed from $\{x_i\}_{i \in C}$ and $f(x_1, x_2, \ldots, x_n)$. Note that here we do not require that $|C| < n/2$.

The protocol starts by each party sharing each of his input bits with all other parties, in a way that exactly n parties are needed in order to reconstruct the input bit. More specifically, to share each of his bits, denoted b, the ''owner'' chooses at random n bits $b_1, b_2, \ldots, b_n$ satisfying $b = \sum_{i=1}^{n} b_i$. Next, the ''owner'' uses the public-key of each party P_i to secretly send to P_i the corresponding piece of b (i.e. party P_i gets $E_i(b_i)$, where E_i is P_i's public encryption function).

At this point, the parties hold pieces allowing them to obtain the values of all the input lines to the boolean circuit. Our purpose is to allow the parties to hold pieces allowing to obtain the value of the output line of the circuit. To this end, the parties will scan the circuit sequentially in the predefined order, generating pieces for the output value of a gate from the pieces of the input lines values. We distinguish between two cases in computing the output value of a gate:

1) The next output line is obtained by adding the constant 1 to the value of some previous line (say line L). In this case, one of the parties (say the first party) adds the constant 1 to his piece of line L, resulting in his piece of the current line. All other parties let their piece of line L be their piece of the current line.

2) The next output line is obtained by multiplying the values of two lines, denoted L_1 and L_2. Let c_i be the ith piece of line L_1 (held by P_i), and d_i be the ith piece of line L_2. We need to compute n pieces of the output line, namely n bits $\{b_i\}_{i=1}^{n}$, such that $b \triangleq \sum_{i=1}^{n} b_i = (\sum_{i=1}^{n} c_i) \cdot (\sum_{j=1}^{n} d_j)$, and each P_i knows **only** b_i.

   Define $b_{i,j}$ so that for each i, $b_{i,i} = c_i \cdot d_i$ and for every $i \neq j$, $b_{i,j} + b_{j,i} = c_i \cdot d_j + c_j \cdot d_i$, and let $b_i = \sum_{j=1}^{n} b_{i,j}$.

   Note that

$$c \cdot d = (\sum_{i=1}^{n} c_i) \cdot (\sum_{j=1}^{n} d_j) = (\sum_{i=1}^{n} c_i \cdot d_i) + \sum_{1 \le i < j \le n} (c_i \cdot d_j + c_j \cdot d_i) = \sum_{i=1}^{n} b_{i,i} + \sum_{1 \le i < j \le n} (b_{i,j} + b_{j,i}) = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{i,j} = \sum_{i=1}^{n} b_i = b.$$

The idea is to let each party P_i compute $b_{i,i}=c_i \cdot d_i$ by himself, and each pair $i \neq j$ of parties execute a **two-party** protocol for *privately computing* $b_{i,j}$ and $b_{j,i}$. This means that:

(1) Party P_i ends with $b_{i,j}$ and party P_j ends with $b_{j,i}$, so that $b_{i,j}+b_{j,i}=c_i \cdot d_j+c_j \cdot d_i$.

(2) The protocol does not leak any further knowledge neither to the participants not to an outside listener.

This protocol, denoted TPIP (Two Party Inner Product), is defined and discussed in section 4.2. Once this two party protocol is executed between each pair of parties, each party, P_i, knows exactly $\{b_{i,j}\}_{j=1}^{n}$ and can thus let $b_i = \sum_{j=1}^{n} b_{i,j}$ be his piece of the current line.

When the output value of $g_{\max}$ is computed, each party sends its piece of the output value to each of the other parties and the output of the function is computed by each party.

The correctness of the output of the protocol is due to the correctness of the output value of each gate. (recall that the parties are semi-honest). The privacy stems from the fact that for any line value b, any proper subset of the n pieces of b does not add any knowledge about the value of b. Thus (assuming the correctness of such a protocol for two parties) the above protocol *privately computes f*. (In other words, we reduced the semi-honest protocol problem to a problem of privately computing a specific function between two parties.) □

4.2 The Two Party Protocol and Oblivious Transfer

The formal specification for the Two Party Inner Product modulo 2 (TPIP) protocol is:

Table 1: **TPIP protocol specification**

	party A	party B
input	a_1, a_2	b_1, b_2
output	c_1	c_2
	s.t. $c_1+c_2=a_1 \cdot b_1+a_2 \cdot b_2$.	

Namely, given the inputs, the protocol will allow parties A and B to compute their specified outputs, without revealing any other information (e.g., the other party's output). In other words, whatever a party can efficiently compute after the protocol has terminated, aside from its output, can be efficiently computed from its initial input alone. (Recall that all the parties are at least semi-honest).

The protocol will use an Oblivious Transfer scheme between two parties. In the OT_1^k scheme, party S has k secret bits, revealing one of them to R, in a way that R knows only one secret and S does not know which secret R knows. Formally, the specification is:

Table 2: OT_1^k **protocol specifications**		
	party S	party R
input	$s_1 ... s_k$	$i \in \{1..k\}$
output	-	s_i

We show a TPIP protocol, using an OT_1^4 (1-out-of-4 Oblivious Transfer) scheme:

Table 3: **TPIP protocol using** OT_1^4		
	party A	party B
input	a_1, a_2	b_1, b_2
The ''reduction'' part	chooses $c_1 \in_R \{0,1\}$. computes $s_{00} \leftarrow c_1$ $s_{01} \leftarrow c_1 + a_2$ $s_{10} \leftarrow c_1 + a_1$ $s_{11} \leftarrow c_1 + a_1 + a_2$.	computes $i \leftarrow b_1 \circ b_2$
Applying OT_1^4	-	s_i
output	c_1	$c_2 \leftarrow s_i$

Note that in each of the four cases (s_{00} , s_{01} , s_{10} , s_{11}) the value $s_{b_1 \circ b_2}$ satisfies $c_1 + s_{b_1 \circ b_2} = a_1 \cdot b_1 + a_2 \cdot b_2$, thus the output $c_2 = s_i$ is correct.

It remains to be shown that the TPIP protocol is private. This is formally stated and proven in Appendix 2.

We now show an OT_1^k protocol, using a one-way permutation with a trapdoor (assuming its existence), along with its hard-core bit. Namely, let G be a generator that on input 1^n generates a pair $(v, t(v))$. For each such pair, let $f_v : D_v \to D_v$ and $b_v : D_v \to \{0,1\}$. Informally, f_v and b_v satisfy:

1. Given v and $x \in D_v$, both $f_v(x)$ and $b_v(x)$ are efficiently computable.
2. Given $t(v)$ and $x \in D_v$, both $f_v^{-1}(x)$ and $b_v(f_v^{-1}(x))$ are efficiently computable.
3. No algorithm can, given only v and $x \in D_v$, efficiently compute $f_v^{-1}(x)$, nor predict $b_v(f_v^{-1}(x))$.

The protocol is the following:

Table 4: OT_1^k **protocol**

	party S	party R
input	$s_1 ... s_k$	$i \in \{1..k\}$
	computes $(v, t(v)) \leftarrow G(1^n)$ sends $v \rightarrow$	chooses $x_1 ... x_k \in_R D_v$
		computes $\vec{y} = y_1 ... y_k$ where $y_j \leftarrow \begin{cases} x_j & j \neq i \\ f_v(x_i) & j = i \end{cases}$ sends $\leftarrow \vec{y}$
	computes $\vec{z} = z_1 ... z_k$ where $z_j = s_j \oplus b_v(f_v^{-1}(y_j))$. sends $\vec{z} \rightarrow$	
output	-	$s_i \leftarrow z_i \oplus b_v(x_i)$

Note that

$$z_i = s_i \oplus b_v(f_v^{-1}(y_i)) = s_i \oplus b_v(f_v^{-1}(f_v(x_i))) = s_i \oplus b_v(x_i)$$

thus

$$z_i \oplus b_v(x_i) = s_i \oplus b_v(x_i) \oplus b_v(x_i) = s_i$$

and the output of party R is correct.

It remains to be shown that the protocol is private, assuming that f is one-way. Informally, it is to be shown that S cannot efficiently compute which secret, s_i, R knows, and R cannot efficiently predict any s_j, $j \neq i$.

The formal statement and proof of these claims can be found in Appendix 1.

5 From Semi Honest to Faulty Parties

Proposition 2. There exists an (explicitly given) transformation from a protocol that *privately computes* f in the presence of semi-honest parties, to a protocol that *privately computes* in the presence of less than $\frac{n}{2}$ faulty parties.

Proof. We use the notion of *verifiable secret sharing* (VSS). This is a protocol which allows a *dealer* to distribute to n *receivers* n pieces of a secret s such that the following three conditions are satisfied (where $1 \leq l < u \leq n$):

1. It is infeasible to obtain any knowledge about the secret from any l pieces.
2. Given any u pieces the entire secret can be easily computed.
3. Given a piece it is easy to verify that it belongs to a set satisfying condition 2. In other words, the secret is recognizable, and the pieces should be verifiable as authentic.

A solution for this problem in which $u = l + 1$ is presented in Appendix 3. We will use VSS with $l = \lceil \frac{n}{2} - 1 \rceil$ and $u = \lfloor \frac{n}{2} + 1 \rfloor$, and as there are less than $\frac{n}{2}$ faulty parties and more than $\frac{n}{2}$ honest parties, only the honest parties will be able to retrieve a secret (which they will do when they discover a faulty party).

The transformation gives a protocol which consists of two stages. In the first stage each party commits itself to its input value, and generates a ''certified random input''. In the second stage a ''certified execution'' of the protocol takes place.

The first stage is done in the following method: First, each party uses VSS to share its private input with all other parties. Then, each party P_i chooses a random string r_{ij} for each $1 \leq j \leq n$. It then shares r_{ij} with all other parties using VSS. If some party P_i fails to do so, then for every j, the string r_{ij} is assigned $\vec{0}$. Now, each P_i broadcasts r_{ij} for each $j \neq i$, and proves that r_{ij} indeed corresponds to the value committed in the VSS. (P_i may prove it by revealing its random choices for the VSS). If it fails to do so this value can be obtained from the VSS messages of P_i. Finally, P_i computes the value $\bigoplus_{k=1}^{n} r_{ik}$, to be used as its random tape.

Note that:

- Party P_i is committed to $\bigoplus_{k=1}^{n} r_{ik}$ as its random input: At any time after this stage the random input of P_i can be reconstructed by the majority of the honest parties by reconstructing $\{r_{ik}\}_{k=1}^{n}$, since they were shared using the VSS method.
- In order to continue on behalf of a faulty party that decides to quit, it is sufficient for the majority of the honest parties to reconstruct his random tape and his local input.
- In this order of commitments (namely first the local input and then the random tape), in any stage a party decides to quit, either he has not learned any additional information, or the honest parties can

already continue on his behalf. However, if the commitments were done in the reverse order (namely first the random tape and then the local input), then a faulty party could decide to quit *after* he has seen his certified random tape (thus gaining additional information to his local input and output), but *before* the honest parties can reconstruct his local input. Then, we might not be able to show that the honest parties can continue on his behalf.

The second stage consists of a ''certified execution'' of the original protocol: assume P_i is to send message m to P_j. The first step is that P_i sends $E_j(m)$ for a public-key encryption function E_j, in order to prevent any other (faulty) party from listening to that message. Then, it is to be proved (in zero-knowledge) that m is indeed a message that P_i is supposed to send at this stage. However, recall that the messages sent in the original protocol (and hence their encryptions) are computed in polynomial-time when given the private and random input of their sender and all the messages it has received so far. Therefore the following statement is in NP:

'' *There exist a string* r_i *which was certified (according the aforementioned scheme) using the communication done with party* P_i *in the first stage of the transformation, and an input* s_i *which was shared (using VSS) by the communication done earlier with* P_i*, and a series of coin tosses* q*; such that the message that party* P_i *is sending now to* P_j *is a legitimate encryption* $E_{j,q}(m)$ *of the proper message* m *(according to the protocol) for these input, random tape, and the encrypted messages received so far, and that message is indeed to be sent to* P_j''.

As an NP predicate, its validity can be proven in zero-knowledge. If P_i fails to do so, the majority of non-faulty parties can detect this, reconstruct his private input, and continue on his behalf.

Thus, faulty parties can suspend the execution of the protocol only in the first stage, but then they do so obliviously of the private inputs of the non-faulty parties and their random input, and furthermore they will be detected. On the other hand, the faulty parties cannot gain any knowledge of the input of some non-faulty party, as only a majority of the parties can compute it given the pieces of the secret that it shares. □

Appendix 1: Implementation of OT_1^k.

Theorem 2. The protocol specified in Table 4 *privately computes* the 1-out-of-k Oblivious Transfer, as defined by Table 2.

Proof. It is to be shown that the three conditions in Definition 1 hold. Condition 1, namely the correctness of the output if all the parties are honest, was shown in section 4.2. Condition 2 is trivially satisfied since

both parties are semi-honest, thus the $\perp$ output never occurs.

In order to show condition 3, we have to consider every set T of honest parties. We have two interesting cases: $T = \{R\}$ and $T = \{S\}$. They are shown in Claims 1 and 2, correspondingly. ■

Claim 1**.** Let π be the presented protocol, let $\vec{s}$ be the k-ary vector of secret bits that is the input to party S, and let $i \in \{1...k\}$ be the input to party R. Then, for every efficient algorithm S' there exists an efficient algorithm M such that

$$M(\vec{s}) \;=\; (ex_{\{R\}}(\pi, S', (\vec{s}, i)))_{\{S\}}.$$

Recall that the right side of the equation denotes the output of S' on input $\vec{s}$, after running the protocol with an honest party R whose input is i. Note that the above is an equation between random variables.

Proof. It suffices to show that it is possible to (efficiently) produce the read-only tapes of S' (namely, the private input, the random input and the received communication) in a distribution identical to that in the above execution of the protocol (i.e., $ex_{\{R\}}(\pi, S', (\vec{s}, i))$). Algorithm M will thus run S' with those ''produced'' tapes as its read-only tapes and output the output of S'.

We show how the read-only tapes are produced: the local input of S' (i.e. $\vec{s}$) is the input to M, and the random tape of S' can also be generated by M since it is a sequence of independent, unbiased coin tosses. The only communication received by S during the protocol is the vector $\vec{y}$ that party R sends in the second stage of the protocol. However, since R chooses $x_i \in_R D_v$ for every $i \in \{1...k\}$ (for v generated by S), and since $f: D_v \to D_v$ is a **permutation** for every v, then $f_v(x)$ is also distributed uniformly in D_v. Therefore, the vector $\vec{y} = (x_1, ..., x_{i-1}, f_v(x_i), x_{i+1}, ...x_k)$ that party S receives in the second stage of the protocol, is a vector of k random variables distributed uniformly over the domain. Thus, algorithm M will generate the ''permutation-index'' v (in the same way that S does) and then choose $\vec{y}' \in_R (D_v)^k$. Thus $\vec{y}'$ is distributed exactly as $\vec{y}$, for every pair $(i, \vec{s})$. □

Claim 2**.** Let π be the presented protocol, let $\vec{s}$ be the k-ary vector of secret bits that is the input to party S, and let $i \in \{1...k\}$ be the input to party R. Then, for every efficient algorithm R' there exists an efficient algorithm M such that

$$M(i, s_i) \quad (ex_{\{S\}}(\pi, R', (\vec{s}, i,)))_{\{R\}}.$$

Recall that the right side of the approximated equation denotes the output of R' on input i, after running the protocol with an honest party S whose input is $\vec{s}$, and that the $\approx$ sign is a relation between two random variables, meaning ''polynomially indistinguishable''.

Proof. As in Claim 1, it suffices to show that it is possible to (efficiently) produce the read-only tapes of R' (namely, the private input, the random input and the received messages) in a distribution indistinguishable from that in the above execution of the protocol (i.e., $ex_{\{S\}}(\pi, R', (\vec{s}, i))$).

We show how the read-only tapes are produced: again, the local input of R' (i.e., the index i) is an input to M, and the random tape of R' can also be generated by M. The messages received by party R during the

execution of the protocol are v and $\vec{z}$, where v is the ''index'' of the one-way permutation, and $\vec{z}$ is the vector sent in the third stage of the protocol. Thus, M will compute $(v,t(v)) \leftarrow G(1^n)$ and give v to R'. Then, upon receiving $\vec{y} \in (D_v)^k$ from R', algorithm M will compute $\vec{z'} = (\sigma_1,...\sigma_{i-1}, s_i \oplus b_v(f_v^{-1}(y_i)), \sigma_{i+1},..., \sigma_k)$ where each $\sigma_j \in_R \{0,1\}$, and set $\vec{z'}$ as the message received by R'. (Namely, the ith bit of $\vec{z'}$ is identical to that of $\vec{z}$ and all the other bits are chosen at random in uniform distribution. Note that f_v can be efficiently inverted by M since it has $t(v)$.)

It remains to be shown that $\vec{z'} \equiv \vec{z}$, given $\vec{x} = (x_1...x_k) \in_R (D_v)^k$, and $\vec{y} = (x_1,...,x_{i-1}, f_v(x_i), x_{i+1},...x_k)$ and v (as this is what is known to R').

This can be shown using a result that was shown in the discussion of hard-core bits, namely that $\sigma \equiv b_v(f_v^{-1}(y))$, given only y and v, where $\sigma \in_R \{0,1\}$, and $y \in_R D_v$. Details follow: let z_j denote the jth bit of $\vec{z}$ and z'_j denote the jth bit of $\vec{z'}$. Thus, $z_i = z'_i$, and for $j \neq i$, we have $z_j = s_j \oplus b_v(f_v^{-1}(y_j))$ and $z'_j = \sigma \in_R \{0,1\}$. Therefore, if $\vec{z}$ and $\vec{z'}$ are polynomially distinguishable, it is then possible, using the hybrids method, to show that there exists an algorithm that efficiently distinguishes between $\sigma \in_R \{0,1\}$ and $b_v(f_v^{-1}(y))$, thus contradicting the above result.

Thus, we have produced an environment for R' that is indistinguishable from an execution of π with an honest party S whose input is $\vec{s}$. □

Appendix 2: Implementation of TPIP using OT_1^4.

Theorem 3. The TPIP protocol presented in Table 3 *privately computes* the TPIP function as defined in Table 1.

Proof. As in Theorem 1, it is to be shown that the three conditions in definition 1 hold. Due to similar arguments, conditions 1 and 2 are clearly satisfied. In order to show condition 3, we again have to consider every set T of honest parties. In each case, it is enough to show an algorithm that produces an environment indistinguishable from the expected one. The two interesting cases are:

1. $T = \{B\}$. It is to be shown that for every efficient algorithm A' there exists an efficient algorithm M_A such that

$$M_A(a_1,a_2,c_1) \equiv (ex_{\{B\}}(TPIP,A',(a_1,a_2;b_1,b_2)))_{\{A\}}, \qquad \text{where} \quad c_1 \in_R \{0,1\}.$$

Algorithm M_A will use algorithm M guaranteed by Claim 1 (M receives as input a vector of secret bits and produces an output with the same distribution as that of the output of party S). Namely, M_A will, given a_1,a_2,c_1 produce the four secrets $s_{00}...s_{11}$ exactly as done in table 3, and run M on that input. Since no conversation is made other than in the OT_1^4 part, the output of M_A and that of A' are of the same distribution, and the claim holds for this case.

2. $T = \{A\}$. It is to be shown that for every efficient algorithm B' there exists an efficient algorithm M_B such that

$$M_B(b_1,b_2,c_2) \equiv (ex_{\{A\}}(TPIP,B',(a_1,a_2,b_1,b_2)))_{\{B\}}, \qquad \text{where} \quad c_2 \in_R \{0,1\}.$$

Algorithm M_B will use algorithm M guaranteed by Claim 2 (M receives as input an index i and a secret s_i, and produces an output whose distribution is indistinguishable from that of the output of party R). Namely, M_B will, given b_1, b_2, c_2 produce $i \leftarrow b_1 \circ b_2$ and $s_i \leftarrow c_2$ and run M on that input. Since no conversation is made other than in the OT_1^4 part, the distributions of the output of M_A and that of A' are indistinguishable, and the claim holds for this case as well. ■

Appendix 3: Implementing VSS.

Verifiable Secret Sharing:

We now present a protocol solving the problem of Verifiable Secret Sharing.

Problem definition:

VSS is a protocol which allows a *dealer* to distribute to *n receivers n* pieces of a secret *s* such that the following three conditions are satisfied, for some $1 \leq t < n$:

1. It is infeasible to obtain any knowledge about the secret from any *t* pieces.
2. Given any $t+1$ pieces the secret can be easily computed.
3. Given a piece it is easy to verify that it belongs to a set satisfying condition (2). In other words, the secret is recognizable, and the pieces should be verifiable as authentic.

The protocol:

1. The dealer sends (on a broadcast channel) a commitment, $c(s)$, to the secret *s* it wants to share.
2. The dealer chooses at random a polynomial $p(x)$ of degree *t* over the field GF(q) (for some $q > n$), whose free element is *s*. Namely, $p(x) = \sum_{i=1}^{t} a_i x^i + s$, where $a_i \in_R GF(q)$. The dealer then broadcasts $E_1(p(1)), E_2(p(2)), \cdots, E_n(p(n))$, where E_i is a public encryption whose decryption key is known to party P_i.
3. Define the following predicate:

   $\exists p$, a polynomial, such that $\deg(p) \leq t$.

   and

   $\exists s$, such that $c(s)$ is a commitment on *s* (i.e. there exists a sequence *r* of coin tosses s.t. $c(s) = c(r,s)$).

   and

   $\forall i$, the i^{th} component of the message sent in stage 2 is $E_i(p(i))$ (i.e. there is a sequence *r* of coin tosses such that the component sent is $E_i(r, p(i))$).

   Note that this is an NP predicate, and since all NP languages have zero-knowledge proofs the dealer now proves the correctness of the predicate in zero-knowledge.

Correctness of the protocol:

Recall that any polynomial of degree t can be interpolated if, and only if, at least $t+1$ of its values are given. Thus, any t pieces yield only t values of $p(x)$, giving no information about its free element, s. (For every possible value of s there exists a (unique) polynomial that passes through the given t points and whose free variable is s). Thus, condition 1 is satisfied. On the other hand, given $t+1$ values, a unique polynomial of degree $t+1$ is fixed: the polynomial whose values at the $t+1$ points are the given ones. Thus, the secret is the free variable of that polynomial, satisfying condition 2.

The zero-knowledge proof in stage 3 of the protocol allows each party to ensure that it gets a certified piece of the secret, and so satisfies condition 3. As this is a *zero-knowledge* proof, no party can efficiently compute from it anything that is not efficiently computable from the initial information it has.