ELSEVIER

Theoretical Computer Science 191 (1998) 173–183

Theoretical Computer Science

Note

Secure multiparty computations without computers

Valtteri Niemi[a], Ari Renvall[b,*]

[a] *Department of Mathematics and Statistics, University of Vaasa, PL 700, 65101 Vaasa, Finland*
[b] *Turku Centre for Computer Science and Department of Mathematics, University of Turku, 20014 Turku, Finland*

Communicated by A. Salomaa

Abstract

Many simple calculations can be done easier without computers than by using them. We show that the same holds for secure multiparty computations if the function to be computed is simple enough. Our starting point is an observation of Bert den Boer: a multiparty computation of a logical AND-gate can be performed by five simple playing cards. We show that by using a reasonable amount of cards many useful functions can be computed in such a way that each input stays private.

1. Introduction

Increasingly large amounts of computations are taken care of by computers, but other computing methods also seem to survive. A pencil and a piece of paper are still more easily accessible than a computer. It is also clear that many simple calculations can be made much easier without computers.

On the other hand, people often take a prejudiced attitude in regard to the use of computers. By experience they learn that, for instance, automatic teller machines very seldom miscalculate, but the attitude does not totally vanish. This fact is one obstacle that prevents expansion of the practical use of cryptographic protocols. Even if a protocol is provably secure against all cheaters, people are afraid that their own personal computers may not function exactly as they assume them to do.

For example, let us consider a system for computerized elections (see, e.g. [1, 5, 6]). It might be the case that every transmission between the PC of a user and the system is perfectly secure and no information about the voting strategy of the user is leaked

* Corresponding author. E.-mail: ariren@utu.fi.

PII S0304-3975(97)00107-2

out. Still, the user is not easily convinced that all inputs given by the keyboard are not saved into some hidden files where computer experts can read them.

There are also other reasons in favor of constructing computation methods that do not depend on computers: electricity might be cut in a critical moment, computers also break down occasionally, computers are not very cheap, a network of PCs is not easily portable, etc. Sometimes it is also possible to find computing methods that are more convenient than the use of computers.

We investigate secure multiparty computations (see, e.g. [3]). One of their typical features is that the function to be computed collectively by the participants is itself relatively easy to compute, but heavy computations are needed to hide the input of each participant from the others. For instance, assume that a small group wants to take a secret vote in a meeting. Surely, it is usually much easier to construct ballots and a ballot box than to set up a computer voting protocol even if computers are available. Using this traditional method, individual inputs are hidden by shuffling them in a box. Unfortunately, this hiding method works only for a very limited number of functions.

Our purpose is to construct respective physical hiding methods for a larger class of functions. The starting point is an elegant protocol of Bert den Boer [2]: five simple playing cards are used in order to compute the value of logical AND. The secrecy of inputs is essentially guaranteed by "cutting the deck" a couple of times. Although AND is a very simple function, it has several meaningful applications. In addition to the one given in [2], let us mention the following one. Two parties are trying to make a contract (for example, the parties could be an employer and a labor union). If the negotiations do not succeed an arbitrator might be called to propose a compromise. For tactical reasons, the parties do not want to tell immediately (not even to the arbitrator) whether or not they accept the proposal. Instead, a card protocol is used to decide whether or not *both* parties accept the proposal.

We generalize the five-card trick of [2] to cover any Boolean function. The number of cards needed grows in the worst case linearly with respect to the number of gates in the circuit that corresponds to the function. It is not very easy to handle a deck of one thousand cards but several useful functions can be computed with a far smaller deck. The time complexity of the protocol also grows linearly with respect to the number of gates.

In [4], Claude Crépeau and Joe Kilian use similar methods to obtain a solitary card protocol for selecting a random permutation with no fixed points. In fact, their paper includes all the necessary tools for a general protocol for multiparty computations. The difference to our work is that we use a smaller alphabet (i.e. a smaller number of different cards), and our protocol is also more efficient.

One big problem in computing with cards is the fact that it is not easy to make identical copies of a hidden card. Suppose you have several identical cards of each type available. One card is on the table with face down. How to replace this one card with two similar ones without revealing the type of the original card? In the final part of the paper we give a partially successful answer to the problem.

2. The basic protocol

We first consider a two-party cryptographic protocol for evaluating the value of a binary AND-gate. In [2] the situation is illustrated as follows. Alice and Bob have just met and they want to find out whether they have some particular mutual interest. However, they are both reluctant to show their interest unless they know that the other one is also interested. Unfortunately, no third party trusted by both of them is available.

More formally: Alice has a secret bit a and Bob has a secret bit b. They want to find out the conjunction $a \wedge b$ of these bits, but they are not ready to reveal their own secret bits. In other words, if $a = 0$, Alice should not be able to find out b, and vice versa. (Of course, if $a = 1$ then Alice knows that $b = a \wedge b$). How to proceed?

In [2], den Boer gives a protocol to solve the problem physically. He uses a deck of five cards. Two cards are red and three cards are green, for example. Cards with the same color are identical and, moreover, the back side of each card (red and green) is identical. We give a brief description of the protocol.

2.1. *The "five-card trick"*

Initially, both Alice and Bob have one red and one green card. The remaining green card is put face down on a table. Alice then puts her two cards face down on top of the initial green card. The order of her cards (which Bob should not see) is determined by her secret bit a: if $a = 0$ then the green card is put on top, if $a = 1$ then the top card is red. Now Bob puts his cards at the bottom of the stack. Bob uses a complementary rule when determining the order of his cards: the red card at the bottom means that $b = 1$, and the green card at the bottom means $b = 0$.

After the five-card stack is constructed, Alice and Bob take turns "cutting" the cards, i.e. they rearrange the stack with a random cyclic permutation not known to either of them. When both are satisfied, they display the cards. It is easy to see, that the red cards are next to each other, if and only if, $a = b = 1$ (here we have the cyclic order of cards in mind). Moreover, the other three possibilities are indistinguishable, since the corresponding stacks of cards are cyclic permutations of each others. Thus, using this "trick", Alice and Bob can settle their problem without the fear of getting embarrassed.

It is clear that we can compute some other binary Boolean functions analogously. For example, the result of an OR-gate is obtained by just reversing the commitment rule for both Alice and Bob. As a consequence, the result is 0 iff the two red cards are next to each other.

3. Notations

In the following, we consider cards as elements of an alphabet Δ and ordered sets of cards (i.e. decks) as words over Δ. The topmost card is the first letter of the word,

etc. If $w \in \Delta^*$ we denote $w = w_1 \dots w_n$, where $w_i \in \Delta$. We define functions f_i by

$$f_i : \Delta^* \to \Delta^*, \quad f_i(w) = w_{i+1} \dots w_n w_1 \dots w_i \ (i = 1, 2, \dots, n).$$

In other words, if w is a deck of cards, $f_i(w)$ is obtained by moving the i topmost cards to the bottom of the deck. We also adopt the notation $\langle w \rangle = \{f_i(w) \mid i = 1, 2, \dots, n\}$; i.e. $\langle w \rangle$ is the set of cyclic permutations of w.

Our deck of cards consists of two different kinds of cards: green ones and red ones. The cards have two possible states: they are either "face down" or "face up". Therefore, we define two sets: $\Pi = \{G, R\}$ corresponding to the face down cards, and $\Sigma = \{g, r\}$ corresponding to the face up cards. Using this notation we have $\Delta = \Pi \cup \Sigma$.

For variables corresponding to the face down cards we usually use capital letters $X, Y, \dots$, and small letters $x, y, \dots$ are used with face up cards. Similarly, we use variables $U, V, \dots$ (resp., $u, v, \dots$) for a deck of face down (resp., face up) cards. We also denote $\overline{G} = R$, $\overline{R} = G$, $\overline{g} = r$ and $\overline{r} = g$.

Let φ be the function which corresponds to turning a face down card face up: $\varphi(G) = g$ and $\varphi(R) = r$. We extend φ to a morphism $\varphi : \Pi^* \to \Sigma^*$ in an obvious way. The cryptographic assumption we use in our protocols is that computing φ is possible only if all participants co-operate. (Of course, anyone in the possession of cards can view a face down card, but the others will notice it.)

A commitment of a bit 0 is an element $D_0 = GR \in \Pi^2$ and a commitment of a bit 1 is $D_1 = RG \in \Pi^2$. Denote $D = \{D_0, D_1\}$ and define the decryption morphism $\delta : D^* \to \{0, 1\}^*$ by $\delta(D_b) = b$ $(b \in \{0, 1\})$. Hence, a commitment of a bit is a pair of face down cards, where the one card is green and the other card is red. The order of cards determines the bit in question, similarly to the "five-card trick". We use Greek letters $\alpha, \beta, \dots$ to denote commitments.

Finally, we formalize our main cryptographic tool: cyclic shuffling of a deck of face down cards. Cyclic shuffling of a deck W corresponds to choosing a random element from $\langle W \rangle$. Denote this randomized algorithm by F

$$F(W) = f_i(W), \quad \text{where } i \in_R \{1, \dots, |W|\}.$$

We assume that it is possible to compute F in such way that no one learns the index i. (This is acceptable, since, in practice, anyone is able to cut a deck of cards in such a way that the others cannot observe the exact point of the cut. F can then be computed by letting each participant to cut the deck in turns).

To get familiar with our notation, we redescribe the basic protocol for AND, given in the previous section. Although the protocol has only two participants, we found it convenient to introduce also a "middleman" M. Whatever M does, M does it publicly. In other words, both A and B know everything that M knows. For example, if M computes $\varphi(X)$ then both A and B find out the value. But if A or B computes it, the other will not learn it. In practice, either A or B or both of them together may take the role of M.

First, we give a protocol for encrypting secret bits.

Protocol 1. *A bit commitment protocol.*

A's secret input: A bit a.
(1) M gives A two cards $\alpha'' = GR$.
(2) A computes $\alpha' = F(\alpha'')$ and $a' = \delta(\alpha')$.
(3) A outputs $\alpha = \begin{cases} \alpha' & \text{if } a' = a, \\ f_1(\alpha') & \text{if } a' = 1 - a. \end{cases}$

It is clear that in the protocol $\delta(\alpha) = a$ and that B cannot figure out the bit a.

The five-card trick can now be described as follows:

Protocol 2. *A protocol for A and B for computing the conjunction of their secret bits.*

A's secret input: A bit a.
B's secret input: A bit b.
(1) A and B encrypt their secret bits by α and β (using Protocol 1), and give them to M.
(2) M computes $\gamma = F(\alpha G f_1(\beta))$.
(3) M outputs 0, if $\varphi(\gamma) \in \langle rgrgg \rangle$, and M outputs 1, if $\varphi(\gamma) \in \langle rrggg \rangle$.

4. Computing AND, NOT and OR in encrypted form

The goal of this paper is to construct a "card protocol" for any Boolean function. It seems to be a reasonable approach to build such a general protocol from protocols for simple Boolean operators. These basic protocols should produce a commitment of the output bit, so that it can later be used as input without knowing the value of the bit it encrypts.

In some cases the same input is needed several times in the computation. As our physical card commitments are unreusable, we also need a protocol for copying commitments (without viewing the cards). The following quite natural protocol has also been given in [4].

Protocol 3. *A protocol for generating k copies of a bit commitment $\alpha \in D$ without giving away any information on $\delta(\alpha)$.*

M's input: A bit commitment $\alpha = X\overline{X} \in D$.
(1) M constructs a deck $U = (GR)^{k+1}$.
(2) M computes $U' = F(U) = (Y\overline{Y})^{k+1}$ and sets $V = \alpha_1\alpha_2 U_1' U_2' = X\overline{X}Y\overline{Y}$.
(3) M computes $v = \varphi(F(V))$.
(4) M outputs $W = \begin{cases} U_3' U_4' \dots U_{2(k+1)}' & \text{if } v \in \langle grgr \rangle, \\ f_1(U_3' U_4' \dots U_{2(k+1)}') & \text{if } v \in \langle grrg \rangle. \end{cases}$

Theorem 1. *$W = \alpha^k = (X\overline{X})^k$; and no information on $\delta(\alpha)$ is revealed.*

Proof. If $Y = X$ (i.e. if $U' = (X\overline{X})^{k+1}$) then we have $V = X\overline{X}X\overline{X}$. From this it follows that $\varphi(F(V)) \in \langle grgr \rangle$ and in step 4 we choose $W = U_3'U_4' \ldots U_{2(k+1)}' = (X\overline{X})^k$. Correspondingly, if $Y = \overline{X}$ then $U' = (\overline{X}X)^{k+1}$ and $V = X\overline{X}\overline{X}X$. Hence, $\varphi(F(V)) \in \langle grrg \rangle$, and therefore we choose $W = f_1(U_3'U_4' \ldots U_{2(k+1)}') = f_1((\overline{X}X)^k) = (X\overline{X})^k$. Thus, in both cases we have $W = \alpha^k$, as desired.

It remains to show that no information on X (i.e. $\delta(\alpha)$) leaks during the protocol. The only possible source of information is in step 3, where we compute $v = \varphi(F(V))$. But from v we can only conclude whether or not $X = Y$. This still leaves the cases $X = G$ and $X = R$ equally probable, since Y is totally random and it will not be revealed. □

The total number of cards needed in this protocol is $2k+4$, four of which are "free" after the protocol. It should be noted that while constructing k copies, we loose the original commitment. Thus, there is no sense in producing only one copy.

In the following, we assume that all formulae are composed of AND, NOT and OR operators only. We give a protocol for each operator.

NOT

Computing NOT in encrypted form is trivial. Obviously, if $\alpha \in D$, then $\delta(f_1(\alpha)) = 1 - \delta(\alpha)$. In other words, it is enough to change the order of the commitment cards. To preserve the original commitment we need to take first two copies of α, as explained above.

AND

The AND protocol described in Section 3 is not suitable for our purposes: we need the output in encrypted form. In this section we modify the protocol to obtain the desired properties.

Protocol 4. *A protocol for M to evaluate the outcome of an AND gate in encrypted form.*

M's input: Two bit commitments $\alpha = X\overline{X}$ and $\beta = Y\overline{Y}$.

(1) M computes $W = \alpha^2 = (X\overline{X})^2$ and $W' = \beta^2 = (Y\overline{Y})^2$ by using protocol 3.
(2) M constructs a deck $U = W_1W_2GW_2'W_1'W_3W_4GW_4'W_3' = (X\overline{X}G\overline{Y}Y)^2$.
(3) M computes $U' = F(U)$ and $x = \varphi(U_1')$.
(4) If $x = g$ then M sets $U = U'$ and goes back to step 3. (Of course, the topmost card U_1' is first reset face-down; i.e. M computes $\varphi^{-1}(U_1')$).
If $x = r$ then M computes $v = \varphi(F(U_2'U_3'))$.
(5) If $v = gg$ then M outputs $\gamma = U_{10}'U_9'$.
If $v \in \{gr, rg\}$ then M outputs $\gamma = U_7'U_8'$.

Theorem 2. *In the protocol given above, $\delta(\gamma) = \delta(\alpha) \wedge \delta(\beta)$, but no further information on $\delta(\alpha)$, $\delta(\beta)$ or $\delta(\gamma)$ is revealed.*

Proof. Because of the construction of the word U in step 2 we know that $\varphi(U) = u^2$, where $u = \varphi(\alpha G f_1(\beta))$. Thus, also in step 3, $\varphi(U') = u'^2$, where $u' \in \langle u \rangle$. After step 4 we know that $u'_1 = r$. Now we have only four possible cases: $u' = rrggg$ (outcome is 1) or $u' = rgrgg$ (0) or $u' = rggrg$ (0) or $u' = rgggr$ (1). M has no information of which alternative is the correct one. However, we notice that if $u'_2u'_3 = gg$ then $U'_{10}U'_9$ (and $U'_5U'_4$) is a commitment of the outcome. Correspondingly, if $u'_2u'_3 \in \langle gr \rangle$ then $U'_7U'_8$ (and $U'_2U'_3$) is a commitment of the outcome. In step 5 we test which case is in question (by using cards U'_2 and U'_3) and determine the output according to it.

It remains to be shown that no unintended information on the commitments leaks during the protocol. This is quite obvious. After step 4 we know that either $U' = (RGGZ\overline{Z})^2$ or $U' = (RZ\overline{Z}GG)^2$. But, because we do not know whether $Z = G$ or $Z = R$, this amounts to nothing. □

In addition to the four input cards we need eight more cards to perform the protocol: First, we need six cards in taking two copies of α. Four cards are free after the copying – thus, only two more cards are needed while copying β. Again four cards (two red and two green cards) are ready for reuse, and the two green cards are used as middle cards in step 2. After the protocol, 10 of the 12 cards involved are free and two cards encrypt the outcome.

Again, if we need the original commitments later, we should take a copy of them. This can be done before we perform the protocol, but it is more convenient to do it during the protocol. In step 1 we need to take copies of the inputs anyway, so without any trouble we can take one copy more. Then, however, we need four more cards – thus the necessary number of cards increase up to 16.

Our AND protocol is probabilistic. In step 3 we keep shuffling the deck and looking up the color of the topmost card, until the card is red. Because four of the ten cards are red, the expected number of loops is $2\frac{1}{2}$. We have found a solution to avoid this minor problem. However, the solution is much more complicated and the number of necessary cards is 44. We omit the solution here.

OR

Because $p \vee q = \neg(\neg p \wedge \neg q)$, it is easy to modify the AND protocol to obtain a protocol for OR. The only changes are in step 2, where now M constructs $U = (f_1(\alpha)G\beta)^2$, and in step 5, where the final output is reversed.

It would be equally easy to construct a protocol for the Sheffer function NAND. It is, however, more convenient to construct more complicated computations using AND, NOT and OR than by using NAND.

5. Computing any Boolean function

In this section we give a general protocol for computing any Boolean function. The protocol is straightforward: first the participants commit to their secret bits and then

the value of the function (circuit) is evaluated proceeding operator by operator (gate by gate) using the protocols of the preceeding section. The final commitment obtained from the last gate is then revealed. We do not give a proof or a detailed description of the protocol, as they should be obvious after the preceeding section.

Protocol 5. *A protocol to compute any Boolean function composed of AND, NOT and OR gates in encrypted form.*

M's input: A Boolean circuit with S gates and commitments of the N input bits.

(1) M prepares a deck of ten cards $U=(GR)^5$, and additionally for each gate G_i, M reserves a pair of cards $\beta_i = GR$.
(2) M computes the value of each G_i in proper order using the protocols of the preceding section. In order to do this M uses the input commitments, together with β_i and U.
(3) M computes $\delta(\gamma)$, where γ is the commitment obtained from the computation of the final gate.

Obviously, the number of cards needed in this protocol is $2(N+S)+10$. After the protocol we still have a commitment of each input bit and the output bit of every gate. In most cases this is not necessary and hence the actual number of cards needed is usually lower. A better estimation on the necessary number of cards is obtained by counting the maximum number of bits that need to be remembered simultaneously during the protocol. If this number is denoted by P, then we can manage with $2P+10$ cards.

For example, if our function is the $k$ary AND (i.e. $f(b_1,\ldots,b_k)=b_1\wedge\cdots\wedge b_k$) then at each moment we need to remember only two bits. First b_1 and b_2, then $b_1\wedge b_2$ and b_3, then $b_1\wedge b_2\wedge b_3$ and b_4 etc. According to the formula we can manage with 14 cards. It can be shown that, in this special case, only 12 cards are needed (the same number as with ordinary AND with two input bits).

The AND function is an example of a *threshold function*. The value of a threshold function is fully determined by the number of 1s (or 0s) among the input bits. It is quite easy to see that when computing a threshold function with N input bits, only $N+2$ bits need to be remembered at each moment. Thus, N people can make a majority decision using a deck of $2N+14$ cards in such way, that the exact number of *yes* and *no* votes will not be revealed.

5.1. Zero-knowledge with cards

Zero-knowledge proofs are important building blocks in many cryptographic protocols. In such a proof, a prover P convinces a verifier V of the validity a given (mathematical) statement. The characteristic property is that although V gets convinced, she learns nothing of the proof itself. For instance, assume that P knows a satisfying assignment for a Boolean formula. Using a zero-knowledge proof P can convince V that the formula is satisfiable without giving away any information on the

assignment he knows. As a consequence, V cannot later convince any third party of the satisfiability.

Our protocol for secure multiparty computation can obviously be modified to obtain zero-knowledge proofs for satisfiability. P is the only one with secrets, thus he performs all the necessary operations. P applies the protocol to compute the value of the Boolean formula using the satisfying assignment. V participates only by watching.

Unfortunately, the instances where zero-knowledge proofs have any significance are so large that we would need a huge deck of cards. Thus, this property of our protocol is interesting only from a theoretical point of view. Nevertheless, this example shows that even complicated cryptographic protocols can be implemented without massive computing power.

6. How to copy only one card?

The Protocol 3 is a simple method for making copies of two face down cards which are known to be red and green, but the order is not known. An obvious problem related to this is the following: is it possible to copy a single card similarly? This is clearly a more difficult task.

All our protocols are based on the idea of coding information in the order of cards, hence the problem of copying one card is irrelevant in our setting. However, the opportunity to produce copies of a single card seems to be a very useful tool in constructing new protocols. That is the reason why we devote one section to this problem.

We cannot give a complete and fully satisfactory solution to the problem, but our next protocol is a partial solution which may be sufficient in some cases. The disadvantage of our solution is the possibility that some information about the color of the card is leaked out. The probability of a leak can be made arbitrarily small but only at the cost of exponentially increasing the number of cards.

Protocol 6. *A protocol for M to make k copies of one face down card.*

M's input: A card X and a security parameter s.

(1) M constructs a deck $U''=(GR)^{k+1}$ and s decks $V''_{(i)}=(GR)^{2^i}$ $(i=1,\ldots,s)$.
(2) M computes $U'=F(U'')$ and $V'_{(i)}=F(V''_{(i)})$ for $i=1,\ldots,s$.
(3) M constructs decks $V_{(i)}=V'_{(i),1}V'_{(i),3}\ldots V'_{(i),2^{i+1}-1}$ for $i=1,\ldots,s$.
(4) M constructs a deck $W'=XU'_1V_{(1)}\ldots V_{(s)}$ and computes a random permutation W of W'.
(5) M computes $w=\varphi(W)$. If w contains an odd number of green (and red) cards then M outputs $U=U'_4U'_6\ldots U'_{2k+2}$. If w has an even number of green cards then M outputs $U=U'_3U'_5\ldots U'_{2k+1}$.

Theorem 3. *In the protocol, $U=X^k$ and M learns $\varphi(X)$ only with probability* $(1/2)^{s+1}$.

Proof. Because of the construction method of U'' we know that $U' = (Y\overline{Y})^{k+1}$. Hence, it is clear that if $X = Y$ then $U'_3U'_5 \ldots U'_{2k+1} = X^k$; and if $X = \overline{Y}$ then $U'_4U'_6 \ldots U'_{2k+2} = X^k$. Thus, we need to show that $X = Y$, *iff* w has an even number of green cards.

The deck W is composed of the cards X and $U'_1 = Y$ and the decks $V_{(i)}$. Because $V'_{(i)} = (Z_i\overline{Z}_i)^{2^i}$ for some Z_i $(i = 1, \ldots, s)$, $V_{(i)} = Z_i^{2^i}$. This means that each $V_{(i)}$ consists of an even number of cards of the same color. Hence, if $X = Y$ then necessarily w has an even number of both green and red cards. Correspondingly, if $X = \overline{Y}$ then the numbers of green cards and red cards are both odd.

It remains to show that only with a negligible probability M learns anything about $\varphi(X)$. The only possible source of information is in step 5 where M computes $w = \varphi(W)$. Clearly, if w consists of only green or red cards, then the color of X is revealed. This event occurs only if $Y = X$ and each $Z_i = X$. The probability of this is $(1/2)^{s+1}$.

If w has both green and red cards, there is no way to gain any information on $\varphi(X)$. This can be proved using a straightforward induction. For notational convenience we denote $\psi(w) = (|w|_g, |w|_r)$; i.e. $\psi(w)$ is the *Parikh vector* of w telling us the number of gs and rs in w. Because of the random permutation of step 4 all the information leaked out by w is already contained in $\psi(w)$.

If $s = 0$ then the claim is quite trivial, since the only case to consider is $\psi(w) = (1, 1)$ and there are two equally probable cases. Thus, assume that $s > 0$ and denote $\psi(w) = (i, j)$. The deck w consists of 2^{s+1} cards. If $i > j$ then we can conclude that $V_{(s)} = G^{2^s}$. Removing 2^s green cards from w takes us essentially back to the case of $s - 1$ with the Parikh vector $(i - 2^s, j)$. As neither component is zero, the claim follows from induction hypothesis. Obviously, the case where $i < j$ is symmetric, hence assume that $i = j$. There are only two possibilities: either $V_{(s)} = G^{2^s}$ and the rest cards are red, or vice versa. These cases are equally probable and the claim follows again. □

7. Conclusions and further research

We have shown that it is possible to compute any Boolean function by using only red and green playing cards in such way that during the computation each participant learns nothing else but the final output. The number of cards is a linear function of the number of gates in a circuit corresponding to the computed function. Although this rate of growth is mild, it is important to try to minimize the number of cards needed in computing some specific functions that may have practical applications. More efficient protocols might use, e.g. more colors than just two.

Our protocols are based on the assumption that cyclic permutations of a deck are indistinguishable or symmetric. In group-theoretical terms it can be said that we operate in a cyclic *symmetry group*. A possible generalization is to move into *dihedral* groups. Then also a deck and its mirror image are indistinguishable. This case has also a physical interpretation.

Consider a "necklace" containing small identical boxes hanging from a string. Tiny balls of different colors can be put inside the boxes. Now it is possible to rotate the string corresponding to cutting of the deck and we may also turn the necklace upside down which is a new kind of shuffling operation.

References

[1] J. Benaloh, Verifiable secret-ballot elections, Ph.D. Thesis, Yale University, Technical Report 561, 1987.
[2] B. den Boer, More efficient match-making and satisfiability; the five card trick, Proc. EUROCRYPT'89, Lecture Notes in Computer Science, vol. 434, Springer, Berlin, 1990, pp. 208–217.
[3] D. Chaum, I. Damgård, J. van de Graaf, Multiparty computations ensuring privacy of each party's input and correctness of the result, Proc. CRYPTO'87, Lecture Notes in Computer Science, vol. 293, Springer, Berlin, 1988, pp. 87–119.
[4] C. Crépeau, J. Kilian, Discreet solitary games, Proc. CRYPTO'93, Lecture Notes in Computer Science, vol. 773, Springer, Berlin, 1994, pp. 319–330.
[5] V. Niemi, A. Renvall, How to prevent buying of votes in computer elections, Proc. ASIACRYPT'94, Lecture Notes in Computer Science, vol. 917, Springer, Berlin, 1995, pp. 164–170.
[6] H. Nurmi, A. Salomaa, L. Santean, Secret ballot elections in computer networks, Comput. Security 10 (1991) 553–560.