CWI Quarterly Volume 8 (2) 1995, pp. 129 – 146

The Match-making Problem

Jeroen van de Graaf
Université de Montréal
Département d'Informatique et de Recherche Opérationnelle
e-mail: jeroen@IRO.UMontreal.CA

This paper discusses a cryptographic protocol to evaluate an AND-gate such that a party can keep his or her input bit secret from the other party. Such a protocol is of interest, because it can be generalized to any logical circuit for any number of participants. A formal statement of this generalization reads as follows: n participants want to compute together a function $f(x_1, x_2, ..., x_n)$ with x_i being their inputs; nobody wants to reveal information about his or her input except what can be logically be deduced from one's input and the output. The paper contains no new results but provides an illustration of a sub-field of cryptography, and describes several interesting protocols and protocol design techniques.

1. Introduction

Suppose *Alice* and *Bob* meet for the first time. They both want to find out whether they are interested in each other, so both engage in a cryptographic protocol transmitted in infra-red through their watches. The protocol has to be such that if both parties show interest they find out, but if one party isn't he or she cannot find out whether the other person was interested. This problem, called the *match-making problem*, was raised by David Stodolsky, a social scientist with strong interest in privacy protection, when he visited the CWI a few months after the Crypto Course was held.

Personally I never understood how a cryptographic protocol could be of any help in solving the match-making problem. I had visions of *Alice* and *Bob* nervously fumbling their watches, though this is just an implementation problem (some people expect that in the future humans will have an infra-red sensor implanted [33]). But what about one party's "Ha! Just checking! Ciao!" (or the less subtle "Mmm, now that I have come closer I change my mind")? There is no penalty for defecting, as game theorists would say.

But still, the match-making problem was extremely relevant, because it is just a disguised form of an evaluation of an AND-gate in which the input of a party is kept secret, except from what can be inferred from the output. If this problem were solved it could lead to solving the generalized problem of evaluating any Boolean circuit consisting of AND- and NOT-gates. For many encryption schemes it was clear how to implement a NOT-gate, but the AND-gate formed the bottleneck.

Around that time protocols existed for some specific problems, like the millionaires' problem [38] (two millionaires want to find out who is richest without revealing their wealth), how to play poker [36] [14] and how to hold secure elections [12]. These protocols take advantage of special algebraic properties of the encryption schemes used. Also, Zero-Knowledge had just been defined [28], and people were coming up with protocols in which *Alice* can convince *Bob* that she knows a satisfying assignment for any Boolean circuit [7, 8, 10, 23]. (Note that in these kind of protocols only one party provides an input to the circuit, which makes it easier to handle than the match-making problem.)

Around 1987 the first solutions to the match-making problem and its generalizations began to appear, resulting in a true avalanche of papers in subsequent years. To see why, let us first state the generalization of the match-making problem in a more formal way.

Private Multi-Party Computation (PMPC):
n participants want to compute together a function $f(x_1, x_2, ..., x_n)$ with x_i being their inputs. Nobody wants to reveal information about his input except what can be logically deduced from the output, y.

Clearly this problem can be solved if a party trusted by all n participants (like a judge, or notary public) is present: everybody just hands his input to this trusted party, who does the computation and announces the result. The question is to obtain the same functionality without a trusted party, under the assumption that participants are connected through some transmission channel.

The function f can be a randomized function: participants can provide a random string as part of their input. When f takes the exclusive-or (XOR) of n random bits, one provided by each participant, a trusted random bit is obtained when at least one participant is honest. Furthermore, everybody can learn the output string y. Private output can be obtained when f considers the random input string of a participant as a private key and XORs (parts of) the resulting string y using this key.

Observe that all the aforementioned protocols (millionaires, poker, voting) are special instances of PMPC, and many other cryptographic problems can be expressed this way. For instance, in mutual identification [18] two parties want to verify they possess the same string. This can be expressed as a private computation of $f(x_1, x_2) = [x_1 \equiv x_2]$, where $\equiv$ denotes logical identity.

Protocols for PMPC have been studied extensively at the end of the 1980s and have several aliases, like secure (or secret) distributed computation, or oblivious

circuit evaluation. To do justice to all the researchers that contributed to this problem is beyond the scope of this paper; for a more detailed overview see [21]. But let us briefly sketch four issues involved: underlying assumptions, protocol structure, security properties and resources needed.

Any protocol for PMPC needs an *assumption* in order to work; two dishonest parties with unrestricted computational resources cannot engage in a meaningful cryptographic protocol. Broadly speaking, we can distinguish three types of assumptions:

Participants are computational restricted: Protocols for PMPC can be constructed if we assume that all the participants have only restricted computational resources and that computationally hard problems exist. Here the hard problem can be of a specific nature and possess specific algebraic properties that makes the protocol implementation easier; many of the earlier solutions were of this kind [22, 19, 25]. Some protocols used the more general assumption that one-way functions exist [24] (g is a one-way function if g is easy to compute whereas g^{-1} is hard to compute; if $\mathcal{P} = \mathcal{NP}$ then one-way functions cannot exist).

The majority of participants are honest: If we assume that a reliable broadcast channel exists, that each pair of participants has a private communication channel and that at least $\frac{2}{3}n$ (later improved to $\frac{1}{2}n$) participants are honest, then there exist protocols for multi-party computations [9, 4, 35, 3, 1]. All these protocols are based on secret sharing schemes, in which a participants breaks his secret in $n-1$ shares which he gives to the others. These secret sharing schemes have the property that a certain quorum is needed to reconstruct the secret from the shares, but a smaller number of (dishonest) participants can gain no information about the secret.

Participants are connected by error-prone channels: Usually we assume that when two participants communicate, the transmission of bits is error-less: a bit sent by one party equals the bit received by the other. In practice error-correcting codes are applied to guarantee this property. However, we can abandon this assumption and use this property, the fact that a communication channel may lose or invert its bits, in favor of cryptography. For instance, in the theoretical notion of Oblivious Transfer [34, 18] it is assumed that half of the bits transmitted by a party just disappear; the other party receives a "?". It has been proven that Oblivious Transfer is sufficient for obtaining a PMPC protocol [15, 29, 30, 35, 26, 13]. More practically, Oblivious Transfer can be implemented on top of a Noisy Channel [17, 16] or on a Quantum Channel [5].

It turns out that most protocols represent the function f by a Boolean circuit and they often exhibit the following overall *structure*:

Initialization phase: All participants agree on the circuit to be evaluated and on all parameters of the protocol. Some protocols use pre-computations to speed up the computation phase.

Input phase: Each participant provides its input in an encrypted way, so as to facilitate the private computation.

Computation phase: The participants evaluate the gates that constitute the circuit sequentially, and intermediate bits are encrypted in a way that nobody can learn their value.

Revelation phase: The participants decrypt and learn the output bits.

The following *security properties* are of importance in a protocol for PMPC:

Correctness: when all participants are honest, the output of the protocol is the same as the function it is emulating.

Privacy: no coalition of n' participants can learn information about the input of an honest participant if he does not cooperate, except from what can be deduced logically from the coalition's inputs and the output.

Honesty: no coalition of n'' participants can make an honest participant accept an output that is not equal to what it should be.

Fairness: participants should learn y i.e. no participant should be allowed to learn the output and quit, leaving the others without y.

Resilience: this reflects the protocol's ability to complete the computation of f if several participants stop cooperating during the protocol.

Clearly some of these properties are conflicting (like privacy and resilience) and trade-offs have to be made (like choosing the values for n' and n''). Coming up with the right definitions for these security properties under the aforementioned assumptions took much effort, and has not been completely resolved. For instance, defining privacy when the participants are computationally restricted is intrinsically different from the case where they are unrestricted. The fairness problem has been studied early [31] and has been elegantly solved [11], at least theoretically. More recently, people have come up with a general model that tries to encompass all the various properties and assumptions [2, 1, 32]. However, with the advent of quantum protocols these models need more study.

Yet another issue is the amount of *resources* needed [21]. In other words, one can try to express and optimize the number of elementary operation, protocol rounds, messages sent, etc. as a function of the input size, circuit size, the number of participants, the level of security desired, etc.

This concludes the brief overview of viewpoints to study protocols for PMPC. The remainder of this paper will be devoted to introducing one particular protocol to evaluate an AND-gate (taken from [19]). This example was chosen because the encrypted representation of the AND-gate is rather one-to-one. After some necessary number theory has been introduced, several simpler protocols will be explained. Apart from being interesting in themselves, these protocols introduce general protocol design principles that will be useful in explaining the final protocol.

2. The match-making protocol

2.1. Some number theory

For the protocol presented here, we need to recall some elementary facts from number theory. Let P be a prime number. Then define the Legendre-symbol $L_P(a) = a^{\frac{P-1}{2}} \bmod P$. From now on we ignore the non-interesting case that a is an integer multiple of P. It is well-known that L is a homomorphism from the multiplicative group Z_P^* to $\{-1, 1\}$. (Remember that a homomorphism is a function that partially preserves the group operation. In this particular example we have $L_P(a * b) = L_P(a) \star L_P(b)$. Here $*$ denotes multiplication modulo P and $\star$ denotes ordinary multiplication restricted to $\{-1, 1\}$. When a homomorphism is one-to-one it is called an isomorphism; this means that there are different ways to represent two sets, but that they possess an identical group structure. We will follow the usual convention in mathematics of not writing multiplicative algebraic operators explicitly, except when clarity demands so.) Euler proved that $L_P(a) = 1$ iff $\exists x \in Z : x^2 \equiv a \bmod P$, so the homomorphism L partitions the domain Z_P^* in two subsets, called the quadratic residues resp. quadratic non-residues modulo P. We often call x the square root of a, where the modulus (here P, later N) is understood from the context.

From now on we set $N = PQ$, with P and Q both prime. Then the Chinese Remainder theorem implies that the multiplicative group Z_N^* is isomorphic to the direct product of Z_P^* and Z_Q^*. Therefore it is perfectly meaningful to define the function $J : Z_N^* \rightarrow \{-1, 1\}; J_N(a) = L_P(a)L_Q(a)$. Surprisingly, $J_N(a)$, called the Jacobi-symbol of a modulo N, can be computed very efficiently *without knowing the prime decomposition of* N.

In the remainder of this paper we will restrict ourselves to the elements of Z_N^* that have Jacobi-symbol 1, so we write $Z_N^*(+1) = \{a \in Z_N^* : J_N(a) = 1\}$. By definition of J we find that $J_N(a) = 1$ iff $L_P(a) = L_Q(a) = 1$ or $L_Q(a) = L_Q(a) = -1$, so $Z_N^*(+1)$ is partitioned into quadratic residues and quadratic non-residues, denoted as QR_N and QNR_N, respectively.

Now we are ready to state the computational assumption on which the protocol presented here is based. It was first stated and used by Goldwasser and Micali; see [27] for a more formal statement.

*Quadratic Residuosity Assumption(*QRA*):* When N is the product of two primes, there exists no efficient algorithm to distinguish QR_N and QNR_N when the factorization of N is unknown.

Note that an efficient algorithm for factoring N implies an efficient algorithm to distinguish QR_N and QNR_N (compute the Legendre-symbol $L_P(a)$) but the reverse is not known to be true, so QRA is stronger than assuming that factoring is hard. Note also that we have discarded half of the elements of Z_N^*, namely those for which $J_N(a) = -1$. These elements are clearly all non-residues modulo N because either $L_P(a) = -1$ or $L_P(a) = -1$, but they can play no meaningful role in the QRA, since their non-residuosity can be determined without factoring N by computing their Jacobi-symbol $J_N(a)$.

2.2. Representing bits by numbers

The preceding exercise in number theory gives us a natural representation for bits: a 0 corresponds to an arbitrary element in QR_N, and a 1 corresponds to an arbitrary element in QNR_N. Moreover, if z is known to be in QNR_N, a bit $b \in \{0,1\}$ can be encrypted by picking a random $r \in Z_N^*$, and computing $e = E(b,r) = z^b r^2$, all computations modulo N. Note that the product (or quotient) of two encryptions belongs to the same class (QR_N or QNR_N) as the exclusive-or of the two original bits: $E(b_1)E(b_2) \bmod N$ and $E(b_1 \oplus b_2)$ have the same residuosity. Here $\oplus$ denotes exclusive-or, i.e. addition modulo 2.

From now on we suppose that *Alice* knows the prime decomposition of N, and *Bob* doesn't. Note that both can use the encryption function E, but only *Alice* can do the inverse decryption operation, D. Explicitly, to decrypt e, *Alice* computes $L_P(e)$ and re-interprets the result from $\{1,-1\}$ in $\{0,1\}$. So this encryption scheme, known as probabilistic encryption [27], can be used to send messages from *Bob* to *Alice*. It may seem very inefficient, but it has very interesting theoretical properties.

However, throughout this paper we will not use E and D to establish secure communication, but to obtain a bit commitment scheme. This cryptographic primitive can be explained using an analogy with paper and envelopes. In the first step of a bit commitment *Alice* (say) writes a 0 or a 1 on a piece of paper, puts the paper inside an envelope, seals the envelope and puts it on the table. In the second step *Alice* opens the envelope and shows the number written on the paper to *Bob*. The point to observe is that once the envelope is sealed, *Alice* cannot change her mind on the bit, but *Bob* does not yet know the bit.

Surprisingly, extremely powerful protocols can be constructed using just this simple primitive. A very simple example is a protocol known as coin-flipping by telephone. As the story goes [6], *Alice* and *Bob* are getting divorced and want to divide their common belongings by flipping a coin. They only speak over the telephone (or communicate by email) but they do not trust each other. As can be easily verified, the following protocol resolves their problem.

PROTOCOL 1 (HONESTCOINFLIP)

1: *Alice* chooses a random bit $b \in \{0,1\}$, encrypts it using r, and sends $e \leftarrow E(b,r)$ to *Bob*.

2: *Bob* chooses a random bit $b' \in \{0,1\}$ and sends it to *Alice*.

3: *Alice* sends b and r to *Bob*. The value of the honest coin flip is $b \oplus b'$.

However, before we can use E as a bit commitment scheme we must resolve an important point. A priori *Bob* has no reason to believe that N is indeed of

the form PQ that z is indeed a non-residue modulo N. There are two ways to resolve this point. For the first solution observe that if N has more than two prime factors, than at most $\frac{1}{8}$ of the elements in $Z_N^*$are quadratic residues. So *Bob* will be convinced that N has at most two prime factors if *Alice* shows that a $\frac{3}{8}$ fraction of a large collection of elements from Z_N^*randomly chosen by *Bob* are quadratic residues. Here *Alice* and *Bob* must use PROVEKNOWROOT, to be introduced later.

In addition, *Alice* and *Bob* execute the following protocol from [28] to prove that $z \in \text{QNR}_N$. The notation $v \in_R \mathcal{S}$ means that an element from the finite set $\mathcal{S}$ is picked at random according to the uniform distribution, and its value is assigned to the variable v.

PROTOCOL 2 (PROVEQNR(z, N))

0: *Alice* sends z to *Bob*.

REPEAT k TIMES:

1: *Bob* chooses $b \in_R \{0,1\}$ and $r \in_R Z_N^*(+1)$, computes $e \leftarrow E(b,r)$ and sends e to *Alice*.

2: *Alice* decrypts the message received, $b' \leftarrow D(e)$, and sends b' to *Bob*.

3: *Bob* accepts only if $b = b'$.

ENDREPEAT

The point to observe is that if *Alice* cheats and chooses $z \in \text{QR}_N$, she only receives quadratic residues in step 2 and has to guess b'. So in each round a cheating *Alice* will be caught with probability $\frac{1}{2}$, which reduces to only $\frac{1}{2}^k$ after k repetitions or rounds. Often k is called the *security parameter*: when the amount of work increases linearly the probability for parties to cheat decreases exponentially. This is a generally accepted criterion for a protocol to be "good".

For an alternative way to verify that N and z are of the right form, see the protocol explained in [37].

Given that $z \in \text{QNR}_N$ both *Alice* and *Bob* can invert the value of a bit (i.e. implement a NOT-gate) by multiplying with z: clearly $E(\text{NOT}(b)) \equiv zE(b)$. (In the remainder of this paper $\equiv$ will be used to denote equality modulo N.)

2.3. Showing equality of encryptions

Both parties can easily convince the other of the equality of two encryptions $e_1 = E(b, r_1)$ and $e_2 = E(b, r_2)$ because their quotient (or their product) must be a quadratic residue: $E(b,r_1)E(b,r_2)^{-1} \equiv z^b r_1^2 (z^b r_2^2)^{-1} \equiv (r_1 r_2^{-1})^2$. So one

party can convince the other simply by opening $r_1 r_2^{-1} \pmod N$. Of course, *Alice* can just decrypt two encryptions $e_1 = E(b_1, r_1)$ and $e_2 = E(b_2, r_2)$ created by *Bob* and see whether they are equal or not. But in some situations it can be meaningful when *Alice* is convinced that *Bob* knows this fact too; it proves that *Bob* has constructed e_1 and e_2 using E, instead of having them picked in some sneaky way.

PROTOCOL 3 (PROVEEQUAL(e_1, e_2, N))

0: *Alice* sends $e_1 \leftarrow E(b_1, r_1)$ and $e_2 \leftarrow E(b_2, r_2)$ to *Bob*.

1: *Alice* sends $\tilde{r} \leftarrow r_1 r_2^{-1} \bmod N$

2: *Bob* accepts only if $e_1 e_2^{-1} \equiv \tilde{r}^2$.

Obviously only minor modifications of protocol PROVEEQUAL are needed to make a protocol PROVEUNEQUAL.

2.4. Intermezzo: proving knowledge of a square root modulo N

It is important that in PROVEEQUAL *Alice never* shows the square root on something she has not created herself. Moreover, suppose *Alice* uses the following *insecure* way to show to *Bob* that a number v is in QR_N simply by computing its square root w modulo N (i.e. $w^2 \equiv v$) and sending w to *Bob*. Then with probability $\frac{1}{2}$ *Bob* can find the factorization of N, as follows. *Bob* sends $v \equiv w^2$ to *Alice*, where w is chosen at random. Then *Bob* receives a $\bar{w}$ such that $v \equiv \bar{w}^2$. However, v has two different square roots and with probability $\frac{1}{2}$ he finds $w \not\equiv w'$. (Here we count the square roots w and $-w \equiv (N - w)$ as one root, since one can trivially be computed from the other.) But if *Bob* learns two different square roots w and $\bar{w}$ he can factor N simply by computing the greatest common divisor of $w + \bar{w}$ and N, because $w^2 \equiv \bar{w}^2 \pmod N \Rightarrow (w + \bar{w})(w - \bar{w}) = tN \Rightarrow\ w + \bar{w} = t'P$ or $w + \bar{w} = t''Q$, for $t, t', t'' \in Z$.

So what we really need is a protocol in which *Alice* convinces *Bob* that she *knows* a square root w of v *without showing* w *to Bob*. This is accomplished by the following protocol [28, 20].

The reader can easily verify that if v is not in QR_N, *Alice* has a probability of $\frac{1}{2}$ of being caught in each round, since she has to be prepared for answering $c = 0$ and $c = 1$. The protocol convinces *Bob* because *Alice*'s capacity to answer both challenges simultaneously implies that she indeed knows w. Secondly, notice that under the assumption that factoring is hard neither *Bob* nor a third, eavesdropping party *Eve* can learn w from the protocol, because the only time *Bob* receives something based on w, it has been multiplied by a random

PROTOCOL 4 (PROVEKNOWROOT(v, N))

0: *Alice* has or creates w, computes $v \leftarrow w^2 \bmod N$ and sends v to *Bob*.

REPEAT k TIMES:

1: *Alice* chooses $\bar{w} \in_R Z_N^*$, computes $\bar{v} \leftarrow \bar{w}^2 \bmod N$ and sends $\bar{v}$ to *Bob*

2: *Bob* chooses a challenge bit $c \in_R \{0, 1\}$ and announces c to Bob.

3: If $c = 0$ then *Alice* sends $t \leftarrow \bar{w}$. Otherwise she sends $t \leftarrow w\bar{w} \bmod N$.

4: If $c = 0$ then *Bob* accepts only if $t^2 \equiv \bar{v}$. Otherwise he accepts only if $t^2 \equiv v\bar{v}$.

ENDREPEAT

number, which completely hides its value. This special property of protocol PROVEKNOWROOT called Zero Knowledge, makes it perfectly suitable as a basis for protocols for proofs of identity, in which a client (e.g. a PC, a smart-card) has to identify itself to a host (a mainframe, a bank) [20].

Apart from being of interest in its own right, this small digression allows us to get acquainted with a general mechanism to design protocols, called CUT-AND-CHOOSE, named after the non-cryptographic protocol in which two children have to split a cake: one cuts and the other chooses (the biggest half). The general description of a CUT-AND-CHOOSE is as follows: *Alice* has or creates an object $\mathcal{O}$ (in PROVEKNOWROOT this is w) which has a specific property $\mathcal{P}$ (namely w is a square root modulo N). Let $\mathcal{E}(\mathcal{O})$ be an encryption of $\mathcal{O}$. *Alice* would like to use $\mathcal{E}(\mathcal{O})$, but *Bob* wants to be sure that it satisfies $\hat{\mathcal{P}}$, the property of encrypted objects that corresponds with $\mathcal{P}$. Therefore *Alice* creates a second object $\bar{\mathcal{O}}$, encrypts it and sends it to *Bob*, who can issue two challenges: either to verify that $\mathcal{E}(\bar{\mathcal{O}})$ is indeed the encryption of an object $\bar{\mathcal{O}}$ that satisfies $\mathcal{P}$, or to verify special relations between $\mathcal{E}(\mathcal{O})$ and $\mathcal{E}(\bar{\mathcal{O}})$ that confirm they both satisfy $\hat{\mathcal{P}}$. CUT-AND-CHOOSE is a very powerful technique, dozens of protocols use it and the protocol to evaluate an AND-gate is no exception.

2.5. Choosing white and red balls

Let us return to the encryption function E secure under QRA. By using permutations on several encrypted bits our abilities for making useful protocols increases. Consider the following problem: *Alice* has a vase containing one white ball and $m - 1$ red balls, and *Bob* is allowed to draw one ball. The following protocol emulates this procedure using E. Let both $\mathbf{b} \in \{0, 1\}^m$ and $\mathbf{r} \in (Z_N^*)^m$ denote vectors of length m. Then $E(\mathbf{b}, \mathbf{r})$ denotes encryption of

the entries of $\mathbf{b}$, so its ith entry is of the form $z^{b_i} r_i^2 \bmod N$. Furthermore, if S_m denotes the group of permutations of length m and $\sigma \in S_m$, then $\sigma(\mathbf{b})$, for instance, denotes the vector obtained by permuting the entries of $\mathbf{b}$ according to σ. The composition of permutations is written from right to left.

PROTOCOL 5 (PICKABALL)

0: *Alice* encrypts $\mathbf{b} \leftarrow \langle 1, 0, ..., 0 \rangle \in \{0,1\}^m$ using $\mathbf{r} \in_R (Z_N^*)^m$, she chooses $\sigma \in_R S_m$ and sends $\sigma(E(\mathbf{b}, \mathbf{r}))$ to *Bob*.

REPEAT k TIMES:

1: *Alice* does exactly the same as in step 0 using different random choices for $\bar{r}_i \in_R Z_N^*$, $\bar{\sigma} \in_R S_m$. The result, $\bar{\sigma}(E(\mathbf{b}, \bar{\mathbf{r}}))$, is sent to *Bob*.

2: *Bob* chooses a challenge bit $c \in_R \{0,1\}$ and announces c to *Alice*.

3: If $c = 0$ then *Alice* sends $\bar{\sigma}$ and $\bar{\mathbf{r}}$.
Otherwise *Alice* sends $\tilde{\sigma} \leftarrow \sigma\bar{\sigma}^{-1}$ and $\tilde{r}_i \leftarrow r_{\sigma(i)} \bar{r}_{\tilde{\sigma}(i)}^{-1}$ where $i = 1...m$.

4: If $c = 0$ then *Bob* checks that the m encryptions received in step 1 indeed encrypt the vector $\mathbf{b}$.
Otherwise *Bob* checks that $\tilde{r}_i^2 \equiv r_{\sigma(i)}^2 \bar{r}_{\tilde{\sigma}(i)}^{-2}$ where $i = 1...m$.
ENDREPEAT

5: *Bob* picks $i \in_R \{1, ..., m\}$.

6: *Alice* decrypts the ith component of $\sigma(E(\mathbf{b}, \mathbf{r}))$, i.e.
she sends b_j and r_j, where $j \leftarrow \sigma^{-1}(i)$.

7: *Bob* verifies *Alice*'s decryption, and learns whether he picked the white ($b = 1$) or the red ball ($b = 0$).

Observe the notational conventions used: the barred symbols created in step 1 denote copies of the original objects created in step 0, whereas the symbols with a tilde denote the "quotients". Furthermore note that, apart from steps 5, 6 and 7, the protocol structure of PICKABALL is identical to PROVEKNOWROOT, and so are the reasons why the protocol is secure. Indeed, PICKABALL is another example of a CUT-AND-CHOOSE. Observe that *Bob* only gets to see one entry of $\mathbf{b}$, the other entries remain encrypted. We could obtain a simpler protocol if we were not so strict, but the capacity to hide the other entries of $\mathbf{b}$ is of great value, as we will see. Obviously, this protocol can be easily generalized

to picking one or several balls from a set that contains k white and $m-k$ red balls.

2.6. Blinding of encryptions

Another essential ingredient for the final protocol is the blinding property. Suppose that in an earlier stage *Bob* has received two encryptions e_1 and e_2 from *Alice*. *Bob* can re-encrypt these encryptions using the function R defined as $e' = R(e, b', r') = ez^{b'}r'^2 \bmod N$, where $b' \in \{0,1\}$ and $r' \in Z_N^*$ are chosen randomly. In the envelope analogy it is as if *Bob* is able to change the value of the number written on the paper, even though he does not know what the number is. In addition, *Bob* replaces the old envelope with a new one. Even though this envelope is transparent to *Alice*, she does not know which number she originally put in the envelope.

So if *Bob* sends the two re-encrypted bits e'_1 and e'_2 back to *Alice* in random order, *Alice* is not able to determine the correspondence between the encryptions sent (e_1 and e_2) and received (e'_1 and e'_2) because of the randomly chosen b'_1, r'_1, b'_2 and r'_2, even though *Alice* is able to decrypt. Obviously this technique, called blinding, can be extended from two to any number of encryptions e_i.

In fact, the blinding property allows us to run the protocol PICKABALL with the roles of *Alice* and *Bob* reversed, where *Alice* picks a ball. Steps 0 to 4 are copied from PICKABALL. Thereupon *Bob* will use blinding as has just been introduced. Symbols that have a prime ($'$) will denote objects created by *Bob*.

2.7. Evaluation of an AND*-gate*

After all these preparations the stage is finally set to discuss the protocol for the match-making problem, EVALUATEANDGATE. We will represent an AND-gate by a matrix, so let $\mathbf{T}$ be a 4×3-matrix that represents an AND-gate i.e. $\mathbf{T}$ has rows $\langle 0,0,0\rangle, \langle 0,1,0\rangle, \langle 1,0,0\rangle, \langle 1,1,1\rangle$. We assume that the first column represents *Alice*'s input, the second *Bob*'s input, and the third their common output.

The purpose of the full protocol is that *Alice* and *Bob* together create a double encrypted version of $\mathbf{T}$, denoted $\mathbf{T}^*$, from which they can choose one row that will correspond to their inputs, so they both learn the output bit after having decrypted the output bit for that column. The output column will be encrypted twice, once by *Alice* and once by *Bob*. The input columns are encrypted once; each party encrypts her or his input column.

Steps 0 to 9 of EVALUATEANDGATE and REVERSEPICKABALL are very similar, except that the object to be encrypted is not a vector $\mathbf{b}$, but are 4×3-matrices $\mathbf{W}$, $\mathbf{U}$ and (for *Bob*) $\mathbf{V}$. Here $\mathbf{W}$ denotes $\mathbf{T}$ with some of its entries XORed as to hide the input and output columns using $\mathbf{U}$ and $\mathbf{V}$. We will describe the protocol in three parts: first *Alice* creates an intermediate object

PROTOCOL 6 (REVERSEPICKABALL)

Steps **0-4** from PICKABALL are executed.
Call $\sigma(E(\mathbf{b}))$ as received by *Bob* $\hat{\mathbf{b}}$.

5: *Bob* re-encrypts $\hat{\mathbf{b}}$ using $\mathbf{b}'$ and $\mathbf{r}'$, he chooses $\sigma' \in_R S_m$ and sends $\sigma'(R(\hat{\mathbf{b}}, \mathbf{b}', \mathbf{r}'))$ to *Alice*.

REPEAT k TIMES:

6: *Bob* does exactly the same as in step 5 using different random choices for $\bar{\mathbf{b}}'$, $\bar{\mathbf{r}}'$, and $\bar{\sigma}'$. The result, $\bar{\sigma}'(R(\hat{\mathbf{b}}, \bar{\mathbf{b}}', \bar{\mathbf{r}}'))$ is sent to *Alice*.

7: *Alice* chooses a challenge bit $c \in_R \{0,1\}$ and announces c to *Bob*.

8: If $c = 0$ then *Bob* sends $\bar{\sigma}'$ and $\bar{\mathbf{r}}'$.
Otherwise *Bob* sends $\tilde{\sigma}' \leftarrow \sigma'(\bar{\sigma}')^{-1}$ and $\tilde{r}'_i \leftarrow r'_{\sigma'(i)} (\bar{r}')^{-1}_{\tilde{\sigma}'(i)}$ where $i = 1...m$.

9: If $c = 0$ then *Alice* checks that the m encryptions received in step 6 indeed encrypt the vector $\hat{\mathbf{b}}$.
Otherwise *Alice* checks that $\sigma'(R(\hat{\mathbf{b}}, \mathbf{b}', \mathbf{r}'))$ and $\bar{\sigma}'(E(\hat{\mathbf{b}}, \bar{\mathbf{b}}', \bar{\mathbf{r}}'))$ are both indeed permuted re-encryptions of $\hat{\mathbf{b}}$.

ENDREPEAT

10: *Alice* picks $i \in_R \{1, ..., m\}$.

11: *Bob* undoes his re-encryption by showing b'_j and r_j, where $j \leftarrow (\sigma')^{-1}(i)$.

12: Define $\mathbf{b}^* \leftarrow \sigma'(R(\hat{\mathbf{b}}, \mathbf{b}', \mathbf{r}'))$. *Alice* computes $a \leftarrow D(b^*_i) \oplus b'_j$ and learns whether she picked the white or the red ball.

$\widehat{\mathbf{T}}$ (similar to $\hat{\mathbf{b}}$ in REVERSEPICKABALL), then *Bob* creates $\mathbf{T}^*$, and finally *Alice* and *Bob* open the output bit together.

More precisely, to start *Alice* creates

$$\mathbf{W} = \mathbf{T} \oplus \mathbf{U} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \oplus \begin{pmatrix} u_1 & 0 & u_3 \\ u_1 & 0 & u_3 \\ u_1 & 0 & u_3 \\ u_1 & 0 & u_3 \end{pmatrix}$$

Here u_1 and u_3 are chosen randomly from $\{0,1\}$, and $\oplus$ means addition

modulo 2 of corresponding entries in the matrices. The purpose of u_1 is to hide *Alice*'s input, and of u_3 to hide the output.

Then *Alice* encrypts and permutes the rows of $\mathbf{W}$ and $\mathbf{U}$ according to a permutation $\sigma \in_R S_4$. *Alice* and *Bob* perform a protocol identical to PICKABALL to prove that $\widehat{\mathbf{T}} = \sigma(E(\mathbf{W}))$ is constructed as described above. Here we have dropped the random numbers $r \in Z_N^*$ used in the encryption function E from the notation.

PROTOCOL 7 (EVALUATEANDGATE: CONSTRUCT $\widehat{\mathbf{T}}$)

0: *Alice* creates $\mathbf{W} \leftarrow \mathbf{T} \oplus \mathbf{U}$, chooses $\sigma \in_R S_4$, computes $\sigma(E(\mathbf{W}))$ and $\sigma(E(\mathbf{U}))$, and sends them to *Bob*.

REPEAT k TIMES:

1: *Alice* does exactly the same as in step 0 using different random choices for $\bar{\sigma} \in_R S_4$ and for the $\bar{r}$'s used with E.
The result, $\bar{\sigma}(E(\overline{\mathbf{W}}))$ and $\bar{\sigma}(E(\overline{\mathbf{U}}))$, is sent to *Bob*.

2: *Bob* chooses a challenge bit $c \in_R \{0,1\}$ and announces c to *Alice*.

3: If $c = 0$ then *Alice* sends $\bar{\sigma}$, $\overline{\mathbf{W}}$ and $\overline{\mathbf{U}}$, and shows she encrypted them honestly.
Otherwise *Alice* sends $\tilde{\sigma} \leftarrow \sigma\bar{\sigma}^{-1}$ and all the quotients needed for running PROVEEQUAL and PROVEUNEQUAL on the corresponding entries of $\sigma(E(\mathbf{W}))$, $\sigma(E(\mathbf{U}))$, versus $\bar{\sigma}(E(\overline{\mathbf{W}}))$, $\bar{\sigma}(E(\overline{\mathbf{U}}))$.

4: If $c = 0$ then *Bob* checks that the encryptions received in step 1 indeed encrypt matrices $\overline{\mathbf{W}}$ and $\overline{\mathbf{U}}$ of the proper form.
Otherwise *Bob* checks that $\sigma(E(\mathbf{W}))$ and $\sigma(E(\mathbf{U}))$ correspond to $\bar{\sigma}(E(\overline{\mathbf{W}}))$ and $\bar{\sigma}(E(\overline{\mathbf{U}}))$ respectively.

ENDREPEAT

Now it is *Bob*'s turn to take $\widehat{\mathbf{T}} = \sigma(E(\mathbf{W}))$ and complete the construction of $\mathbf{T}^*$, like in steps 5 to 9 from REVERSEPICKABALL. *Bob* will do to $\widehat{\mathbf{T}}$ something identical to what *Alice* did to $\mathbf{T}$, however, on encrypted bits. *Bob* chooses a matrix $\mathbf{U}' = \langle \mathbf{0}, \mathbf{u}'_2, \mathbf{u}'_3 \rangle$ to hide his input column and the output column. He also chooses an additional matrix $\mathbf{V}' = \langle \mathbf{0}, \mathbf{0}, \langle v'_{13}, v'_{23}, v'_{33}, v'_{43} \rangle \rangle$. Then *Bob*

computes $\mathbf{W}' = \widehat{\mathbf{T}} \otimes E(\overline{\mathbf{U}}') \otimes E(\overline{\mathbf{V}}')$ while using *Alice*'s encryption scheme.

$$\mathbf{W}' = \begin{pmatrix} \boxed{\hat{t}_{11}} & \boxed{\hat{t}_{12}} & \boxed{\hat{t}_{13}} \\ \boxed{\hat{t}_{21}} & \boxed{\hat{t}_{22}} & \boxed{\hat{t}_{23}} \\ \boxed{\hat{t}_{31}} & \boxed{\hat{t}_{32}} & \boxed{\hat{t}_{33}} \\ \boxed{\hat{t}_{41}} & \boxed{\hat{t}_{42}} & \boxed{\hat{t}_{43}} \end{pmatrix} \otimes \begin{pmatrix} \boxed{1} & \boxed{u'_2} & \boxed{u'_3} \\ \boxed{1} & \boxed{u'_2} & \boxed{u'_3} \\ \boxed{1} & \boxed{u'_2} & \boxed{u'_3} \\ \boxed{1} & \boxed{u'_2} & \boxed{u'_3} \end{pmatrix}$$

$$\otimes \begin{pmatrix} 1 & 1 & \boxed{v'_{13}} \\ 1 & 1 & \boxed{v'_{23}} \\ 1 & 1 & \boxed{v'_{33}} \\ 1 & 1 & \boxed{v'_{43}} \end{pmatrix}$$

Here the boxes represent the fact that we are dealing with bits encrypted using E, while $\otimes$ represents multiplication modulo N of corresponding matrix entries.

Bob will choose a permutation σ' and send $\mathbf{T}^* = \sigma'(R(\mathbf{W}'))$, $\sigma'(E'(\mathbf{U}'))$ and $\sigma'(E'(\mathbf{V}'))$ to *Alice*. E' denotes an encryption function that only *Bob* can decrypt. *Bob* must convince *Alice* that he has constructed $\mathbf{T}^*$ honestly, and to that end they execute a protocol identical to step 5-9 of ReversePickABall.

Now let us briefly summarize what $\mathbf{T}^*$ looks like: both parties have permuted the rows of $\mathbf{T}$ using σ and σ'; the output column is hidden by each party (by $\mathbf{U}$ and $\mathbf{U}'$); and on top of that also blinded (by $\mathbf{V}'$). The input columns of $\mathbf{T}^*$ are only hidden by one party, the owner. This enables the parties to find the output of the AND on input x_A from *Alice* and x_B from *Bob*.

In fact, EvaluateAndGate implements much more than the simple match-making protocol:

- Any logical gate can be implemented using this protocol.
- Any number of participants can perform this protocol, since any party can join in and play the role of *Bob*.
- Observe that $\mathbf{V}'$ is indispensable; without it *Alice* is able to find out which permutation σ' *Bob* has used. EvaluateAndGate can be used with alternative choices for the encryption functions E, in particular encryption functions that *Bob* can decrypt. In this case *Alice*, instead of *Bob*, needs to encrypt the bits of the output column separately using a matrix $\mathbf{V}$ similar to $\mathbf{V}'$. For more details, see [19].
- The protocol can be extended to evaluate any logical circuit consisting of many gates, T_i. The only restriction is that when T_1 is connected to T_2, that the inversion bit used to hide the output column of T_1 be identical to the inversion bit of the corresponding input column of T_2. This is exactly the reason why u_3 and u'_3 were introduced: they can hide intermediate results when a circuit is evaluated. In the case only one gate is evaluated they are redundant. Again, for more details, see [19].

PROTOCOL 7 (EVALUATEANDGATE: CONSTRUCT $\mathbf{T}^*$)

5: *Bob* creates $\mathbf{W}' \leftarrow \widehat{\mathbf{T}} \otimes E(\mathbf{U}') \otimes E(\mathbf{V}')$, chooses $\sigma' \in_R S_4$, computes $\sigma'(R(\mathbf{W}'))$, $\sigma'(E'(\mathbf{U}'))$, $\sigma'(E'(\mathbf{V}'))$, and sends them to *Alice*.

REPEAT k TIMES:

6: *Bob* does exactly the same as in step 0 using different random choices for $\vec{r}_i' \in_R Z_N^*$, $\bar{\sigma}' \in_R S_4$. The result, $\bar{\sigma}'(R(\overline{\mathbf{W}}'))$, $\bar{\sigma}'(E'(\overline{\mathbf{U}}'))$, $\bar{\sigma}'(E'(\overline{\mathbf{V}}'))$, is sent to *Alice*.

7: *Alice* chooses a challenge bit $c \in_R \{0,1\}$ and announces c to *Bob*.

8: If $c = 0$ then *Bob* sends $\bar{\sigma}'$ and $\overline{\mathbf{W}}'$, $\overline{\mathbf{U}}'$ and $\overline{\mathbf{V}}'$, and shows he encrypted them honestly.
Otherwise *Bob* sends $\tilde{\sigma}' \leftarrow \sigma'(\bar{\sigma}')^{-1}$ and all the quotients needed for running PROVEEQUAL and PROVEUNEQUAL .

9: If $c = 0$ then *Alice* checks that the encryptions received in step 1 indeed encrypt matrices $\overline{\mathbf{W}}$, $\overline{\mathbf{U}}$ and $\overline{\mathbf{V}}$ of the proper form.
Otherwise *Alice* checks that $\sigma'(E(\mathbf{W}'))$, $\sigma'(E'(\mathbf{U}'))$ and $\sigma'(E'(\mathbf{V}'))$ correspond to $\bar{\sigma}'(E(\overline{\mathbf{W}}'))$, $\bar{\sigma}'(E'(\overline{\mathbf{U}}'))$ and $\bar{\sigma}'(E'(\overline{\mathbf{V}}'))$ respectively.

ENDREPEAT

ACKNOWLEDGMENTS

I would like to thank Claude Crépeau and Alain Tapp for several discussions related to this paper, and an anonymous referee for many valuable suggestions. Also I would like to thank all the people who gave lectures at the Crypto Course; their effort made it a wonderful event. Special thanks are due to David Chaum without whom it would have never happened (and without whom I never suspected that Amsterdam has so many fancy restaurants).

PROTOCOL 7 (EVALUATEANDGATE: OPENING THE OUTPUT BIT)

10: *Alice* announces the indices for the two rows that correspond to $x_A \oplus u_1$ in the first column of $\mathbf{T}^*$.

11: From these two indices *Bob* announces the index of the row that corresponds to $x_B \oplus u_2'$ in the second column. Let us call this index i and let $j \leftarrow (\sigma')^{-1}(i)$.

12: *Bob* opens $E'(u_3')$ and $E'(v_{j3}')$.

13: *Alice* opens $E(u_3)$ and decrypts $d \leftarrow D(t_{i3}^*)$.
Then $x_A \wedge x_B \leftarrow d \oplus u_3 \oplus u_3' \oplus v_{j3}'$.

REFERENCES

1. D. BEAVER (1991). Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2).
2. D. BEAVER (1992). Foundations of secure interactive computing. In J. FEIGENBAUM, editor, *Proc. CRYPTO 91*, pages 377–391. Springer. Lecture Notes in Computer Science No. 576.
3. DONALD BEAVER (1990). Multiparty protocols tolerating half faulty processors. In G. BRASSARD, editor, *Proc. CRYPTO 89*, pages 560–572. Springer-Verlag. Lecture Notes in Computer Science No. 435.
4. M. BEN-OR, S. GOLDWASSER, & A. WIGDERSON (1988). Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, Chicago. ACM.
5. C. H. BENNETT, G. BRASSARD, C. CRÉPEAU, & M.-H. SKUBISZEWSKA (1992). Practical quantum oblivious transfer. In J. FEIGENBAUM, editor, *Proc. CRYPTO 91*, pages 351–366. Springer. Lecture Notes in Computer Science No. 576.
6. M. BLUM (1982). Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137. IEEE.
7. G. BRASSARD & C. CRÉPEAU (1986). Nontransitive transfer of confidence: A perfect zero-knowledge interactive protocol for SAT and beyond. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 188–195, Toronto. IEEE.
8. GILLES BRASSARD & CLAUDE CRÉPEAU (1987). Zero-knowledge simulation of boolean circuits. In A.M. ODLYZKO, editor, *Proc. CRYPTO 86*, pages 223–233. Springer-Verlag. Lecture Notes in Computer Science No. 263.
9. D. CHAUM, C. CREPEAU, & I. DAMGÅRD (1988). Multi-party uncondi-

tionally secure protocols. In *Proc. 20th ACM Symp. on Theory of Computing*, Chicago, 1988. ACM.
10. David Chaum (1987). Demonstrating that a public predicate can be satisfied without revealing any information about how. In A.M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 195–199. Springer-Verlag. Lecture Notes in Computer Science No. 263.
11. Richard Cleve (1990). Controlled gradual disclosure schemes for random bits and their applications. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 573–588. Springer-Verlag. Lecture Notes in Computer Science No. 435.
12. J. D. Cohen & M. J. Fischer (1985). A robust and verifiable cryptographically secure election scheme. In *Proc. 26th IEEE Symp. on Foundations of Comp. Science*, pages 372–382, Portland. IEEE.
13. C. Crépeau, J. van de Graaf, & A. Tapp (1995). Committed oblivious transfer and private multi-party computation. In *Proc. CRYPTO 95*. Springer-Verlag.
14. Claude Crépeau (1987). A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face. In A.M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 239–247. Springer-Verlag. Lecture Notes in Computer Science No. 263.
15. Claude Crépeau (1989). Verifiable disclosure of secrets and applications. In Carl Pomerance, editor, *Proc. EUROCRYPT 89*, pages 181–191. Springer-Verlag.
16. Claude Crépeau (1995). Cryptographic protocols based on the noisy channel. In preparation.
17. Claude Crépeau & Joe Kilian (1988). Weakening security assumptions and oblivious transfer. In S. Goldwasser, editor, *Proc. CRYPTO 88*, pages 2–7. Springer-Verlag. Lecture Notes in Computer Science No. 403.
18. Claude Crépeau & Louis Salvail (1995). Oblivious verification of common string. *CWI Quarterly*, 8:??–??.
19. I. Damgård D. Chaum & J. van de Graaf (1988). Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Proc. CRYPTO 87*. Springer-Verlag.
20. U. Feige, A. Fiat, & A. Shamir (1988). Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94.
21. M. Franklin (1993). *Complexity and Security of Distributed Protocols*. PhD thesis, Computer Science Department, of Columbia University, New York.
22. Zvi Galil, Stuart Haber, & Moti Yung (1988). Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 135–155. Springer-Verlag. Lecture Notes in Computer Science No. 293.
23. O. Goldreich, S. Micali, & A. Wigderson (1986). Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages

174–187, Toronto. IEEE.
24. Oded Goldreich, Silvio Micali, & Avi Wigderson (1987). How to play any mental game, or: A completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 218–229. ACM.
25. Oded Goldreich & Ronen Vainish (1988). How to solve any protocol problem - an efficiency improvement. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 73–86. Springer-Verlag. Lecture Notes in Computer Science No. 293.
26. S. Goldwasser & L. Levin (1990). Fair computation of general functions in presence of immoral majority. In A.J. Menezes and S. A. Vanstone, editors, *Proc. CRYPTO 90*, pages 77–93. Springer-Verlag. Lecture Notes in Computer Science No. 537.
27. S. Goldwasser & S. Micali (1984). Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.
28. S. Goldwasser, S. Micali, & C. Rackoff (1985). The knowledge complexity of interactive proof-systems. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 291–304, Providence. ACM.
29. J. Kilian (1988). Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 20–31, Chicago. ACM.
30. Joe Kilian (1990). *Uses of Randomness In Algorithms and Protocols*, chapter 3. The MIT Press.
31. M. Luby, S. Micali, & C. Rackoff (1983). How to simultaneously exchange a secret bit by flipping a symmetrically biased coin. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 11–22, Tucson. IEEE.
32. S. Micali & P. Rogaway (1992). Secure computation. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 392–404. Springer. Lecture Notes in Computer Science No. 576.
33. Marvin Minsky (1994). Will robots inherit the earth? *Scientific American*, 271(4):109–113.
34. M. Rabin (1981). How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory.
35. T. Rabin & M. Ben-Or (1989). Verifiable secret sharing and multiparty protocols with honest majority. In *STOC89*, pages 73–85.
36. A. Shamir, R. L. Rivest, & L. M. Adleman (1981). Mental poker. In D. Klarner, editor, *The Mathematical Gardner*, pages 37–43. Wadsworth, Belmont, California.
37. Jeroen van de Graaf & René Peralta (1988). A simple and secure way to show the validity of your public key. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 128–134. Springer-Verlag. Lecture Notes in Computer Science No. 293.
38. A.Chi-Chih Yao (1982). Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago. IEEE.