More Efficient Match-Making and Satisfiability

The Five Card Trick

Bert den Boer

Centrum voor Wiskunde en Informatica

Kruislaan 413, 1098 SJ Amsterdam, The Netherlands.

Abstract. A two-party cryptographic protocol for evaluating any binary gate is presented. It is more efficient than previous two-party computations, and can even perform single-party (i.e. satisfiability) proofs more efficiently than known techniques. As in all earlier multiparty computations and satisfiability protocols, commitments are a fundamental building block. Each party in our approach encodes a single input bit as 2 bit commitments. These are then combined to form 5 bit commitments, which are permuted, and can then be opened to reveal the output of the gate.

A Matchmaking Example

Alice and Bob never met before and wish to find out whether they have some particular mutual interest. But naturally each refuses to show interest first, because of the risk of getting an embarrassing "no" from the other. More formally, Alice has a secret bit a and Bob has a secret bit b and a protocol is needed that reveals exactly the logical "AND" of the two bits. Consequently, if Bob's bit is zero he should learn nothing about Alice's bit; if his bit is one, he cannot fail to learn Alice's bit because in that case her bit has the same value as the AND.

One way to achieve the desired protocol is by physical means—more precisely, five cards. The back of all cards are , as usual, the same. The face side of two of the cards are identical, say the two-of-hearts, and the face of the other three are identical, say the two-of-spades.

Initially,each party is given one card of each type and the remaining spade is put face down on the table and Bob then puts his cards face down on top of the initial spade. His secret choice of ordering for the two cards encodes his bit b: heart on top means 1 and the other way round means 0.

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 208-217, 1990.

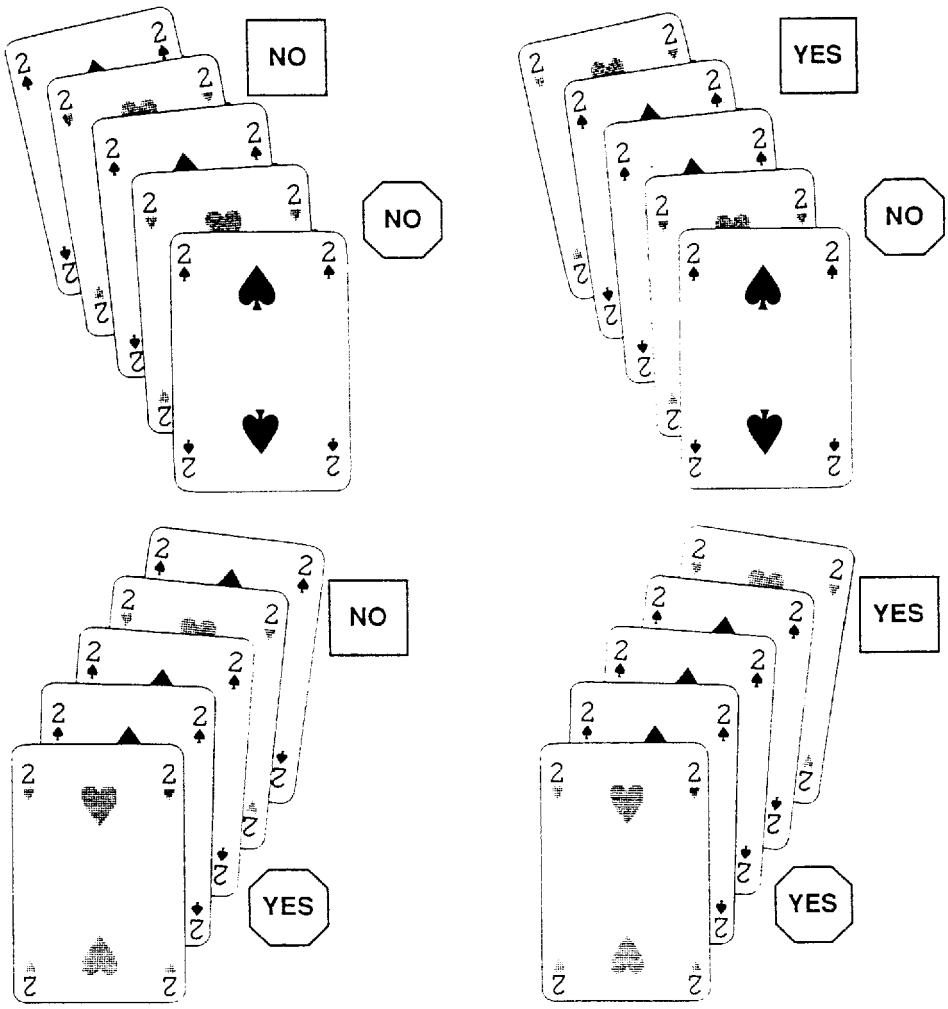

Fig. 1. Match making. If the table had eyes, the table would see five cards: the two of spades in the middle, Bob's two cards on top (the order indicates his choice (in the square)), Alice's two cards at the bottom of the stack (the order indicates her choice (in the octagon)). Three cases are cyclic permutations of each other, so indistinguishable after cutting the cards. The other case is two times yes.

Alice places her cards at the bottom of the stack. The secret order she chooses encodes her bit a, in a way that mirrors Bob: heart on the bottom means *1* and spade on the bottom means *0* (see Fig I)

Then Bob and Alice each take turns "cutting" the cards. The result is a random cyclic permutation known to neither. After each is satisfied that the other does not know the cyclic permutation remaining in the cards, they display the cards on the table in a radial pattern like spokes of a wheel. There are only two distinct results apart from cyclic rotations (see Fig I where the first three cases are cyclic permutations of each other), and they correspond to *AND (a, b)*.

It is easy to see that the two hearts are on consecutive spokes exactly when both bits a and b are *1*.

Introduction

What we just saw is an encoding of two bits into a five bit vector by putting the two bits, their inversion, and an extra zero bit in a certain order and applying a cyclic permutation on this vector. Such a vector is decoded into *1* when it is a cyclic shift of *(1, 0, 0, 0, 1)* or into *0* if it is a cyclic shift of *(0, 1, 0, 1, 0)*. Instead of working with bits we can make sequences of five "blobs". This enables us to make a satisfiability protocol that is more efficient than the similar protocols [BC86]. Our protocol is described in the section: "The main protocol". Satisfiability protocols are as explained in [BC86] ,[BCC] and [GMW86] roughly speaking a way to proof a claim such that the other party cannot prove the same claim by just exploiting the data from the conversation. The above number of five blobs is our gain over the number of twelve blobs used in [BC86].

Before we describe our protocol we exhibit in the section: "Assumptions on Blobs" basicly one extra demand on blobs which we need for efficiency reasons. In order that our idea give rise to a protocol more efficient then the protocol in [BC86] we require that there exist "direct minimal disclosure proofs" for blob equality and blob inequality. In the section: "A Particular Example for a Blob" we suggest a blob which can be made with just one squaring and one computing of a Jacobi-symbol. In the section "Multi party Computations" we exploit our idea for a general multi party computation.

Assumptions on Blobs

In this section we describe requirements for blobs and give a suitable general formula.

For a protocol using blobs to be efficient we require that both creating blobs and opening blobs (showing that a particular blob encrypts a particular bit) are efficient. Bobs are defined, as in [BCC], to satisfy the following: no blob can be opened both as a zero as well as a one (authenticity) and blobs cannot be opened by the opponent (secrecy).

Our blobs should be such that the prover can prove that two blobs hide a different bit (this protocol is called "blob inequality") and also our building block requires that we have a protocol to show that two blobs are encryptions of the same bit without giving information which bit this is (called "blobequality"). It is known that given a protocol for one, a protocol for the other can be constructed.

We also want those protocols to be efficient in the sense that they are non-interactive zero-knowledge with perfect soundness and perfect completeness (see [GMS]). We envisage such a protocol for say blobinequality just by a message of the prover stating that two blobs hide different bits and containing some extra data upon which the verifier can easily verify the statement that the two hidden bits are different. Nonetheless the verifier gets no bias towards which blob hides the zero bit value.

In other words we want a simple protocol where one round gives hundred percent certainty in the involved statement under the assumption that no blob itself can be opened by the prover in two ways (and the verifier has to assume that for any protocol exploiting commitments).

In practice but still in a general setting the way to achieve a bit encoding with a fast protocol for blob equality is the following:

Let G and H be commutative groups,
f a homomorphism from G to H,
$K \in H$, such that the encoder can not find a $c \in G$ such that $f(c) = K$.
The encoding is done in the following way:
take a random element r of G and encode 0 by $f(r)$ and encode 1 by $Kf(r)$.

The reason the encoder cannot find such a c is either because no such value exists or because a suitable value is infeasible to find (because of a cryptographic assumption like the encoder cannot factor or take discrete logs). In the first case the opponent should not be able to distinguish random elements of the image of f with random elements of the coset which includes K because of a cryptographic assumption. In the second case the opponent might know a solution c for the equation $f(c) = K$ but then in his view any blob could be equally likely encode a zero as well as a one. Both cases give different flavors of zero knowledge protocols. In the

second case the encoder is unconditionally protected and the opponent can only be fooled if the encoder could crack the underlying problem during the time of the protocol. In the first case the opponent is unconditionally protected but he could learn the satisfying assignment later on by cracking the underlying problem.

Blob inequality is done by giving the fact that two blobs hide different bits and giving some element s to the verifier then the verifier checks that the product of the blobs is equal to $Kf(s)$. Blob equality is done by giving a element u of G such that $f(u)$ is the quotient of the two blobs. It is easy to see that those techniques meets the demands for blob (in) equality.

The Main Protocol

In this section we exhibit our construction for a satisfiability protocol.

The idea is that the prover P and the verifier V have agreed on some circuit consisting of two-input gates realizing the Boolean expression implied by the assignment. All bit values outside and inside the circuit are probabilisticly encoded by P. The output bit of a gate might be used in more than one subsequent gate. The idea is that P encodes the input bits for the whole circuit (the bits corresponding to his secret: the satisfying assignment) and subsequently all bits inside the circuit. The final bit (which is equal to one) at the outside of the circuit has to correspond with the encodings of the last two bits at the input side of the last gate in the circuit. Now it is sufficient if the verifier is convinced that for each gate the input bits and the output bit satisfy the equation implied by this gate. First we deal with those eight gates which are of degree two. Basicly all eight are similarly dealt with. We show how to deal with the so-called NAND-gate. As is known the whole circuit could be built with NAND-gates with at most five times more gates than in a circuit where all ten possible two-input gates are allowed.

Let us denote the input bits of the NAND-gate a and b and the output bit by c. P encodes the output bit c just by a residue C independent of the encodings A and B of a and b. To prove the consistency of the blobs A, B and C the prover will convince the verifier that the bit vectors $(b\oplus 1, a, 0, b, a\oplus 1)$ and $(c, c\oplus 1, 0, c\oplus 1, c)$ are cyclic permutations of each other. To do this P creates another cyclic shift $(d_0, d_1, d_2, d_3, d_4)$ of this vector. Then P encodes those bits d_i by blobs D_i. Those five blobs are given to the verifier and he has the choice out of two questions.

On the first question the prover gives a residue k modulo five and elements s_0, s_1, s_2, s_3, s_4 of the group G such that the following equations hold:

$$\begin{aligned} D_k &= f(s_0)\, C, \\ D_{(k+1) \bmod 5}\ C &= K f(s_1), \\ D_{(k+2) \bmod 5} &= f(s_2), \\ D_{(k+3) \bmod 5}\ C &= K f(s_3), \\ D_{(k+4) \bmod 5} &= f(s4)\, C. \end{aligned}$$

In case the verifier chooses for the second question it means that the prover has to give a residue m mod 5 and five elements $v_0,\ldots,v_4$ of G such that

$$\begin{aligned} D_m\, B &= K f(v_0), \\ D_{(m+1) \bmod 5} &= f(v_1)\, A, \\ D_{(m+2) \bmod 5} &= f(v_2), \\ D_{(m+3) \bmod 5} &= f(v_3)\, B, \\ D_{(m+4) \bmod 5}\ A &= K f(v_4). \end{aligned}$$

The eight gates of degree two can be described by $gate(a,b) = NAND(a \oplus c, b \oplus d) \oplus e$, where c, d and e are bits. For $c = 1$ we just interchange the positions of a and $a \oplus 1$, for $d = 1$ we just interchange the positions of b and $b \oplus 1$. For $e = 1$ we decimate the positions of all elements in the five bit vector with a factor plus or minus two modulo five.

For the exclusive or gate we take $XOR(a, b) = (a \oplus 1, b, 0, b \oplus 1, a)$. If we decimate the position of the elements in this vector with a factor minus two and if we assume that the middle element is at position zero we get $EQ(a, b) = (b \oplus 1, a \oplus 1, 0, a, b)$. This way we have dealt with all two bit input gates

For each gate several sets of five residues like $(D_0,\ldots, D_4)$ are presented to V. Then all first sets are gathered in one class all second sets in another class etcetera. For each class the verifier V request to see either equivalence of each set of five blobs with either the input side of the involved gate or to see equivalence of each set of five blobs with the output side of the involved gate.

Proof of security: Suppose for a class of sets of residues like $(D_0,..., D_4)$ the prover has to show equivalence of each set of five residues with the input side of the involved gate. The prover can only survive this challenge if either he knows a satisfying assignment and made his blobs according to the protocol or in case P does not know P has made all his sets of residues in this class to be consistent with the inputside of involved gates. In the last case he does not know the equivalence for at least one set of five residues with the output side. So his chance of cheating is 2^{-k} by guessing or predicting the question of V and making the sets of five residues like $(D_0,..., D_4)$ accordingly. Otherwise we have to assume that for each set of five residues like $(D_0,..., D_4)$ in at least one class P can proof equivalence with both input side and output side. We may assume the gate is a NAND-gate then this means that the elements $(b \oplus 1, a, 0, b, a \oplus 1)$ and $(c, c \oplus 1, 0, c \oplus 1, c)$ of $GF(2)^5$ are cyclic shifts of each other.

Now we will have to prove that is precisely the case when $c = NAND(a, b)$. Note that for $a = b = 1$ we do not have two consecutive ones in the cyclic ordering of the bits in the first vector. This can only be the case in the second vector if $c = 0$. For all three other possible values of (a, b) there are two consecutive ones in the first vector and such a vector can be brought in the form $(1, 0, 0, 0, 1)$ by a cyclic shift. This can only happen for $c = 1$.

□

To increase the efficiency more efficient protocols can be made for the special cases (gates of degree one) of the XOR gate and the EQ gate.. We will show it in case the gate is the XOR gate. The first improvement on the protocol is to create blobs E_0 and E_1 and showing equality for the exclusive-or sum of the encoded bits with either the pair A and B or with the pair C and the element $f(1)$(the obvious encoding of zero, where 1 is the unit element of G). The equality of the sum of the bits hidden by E_0 and E_1 and the sum of the bits hidden by A and B is shown by either blobequality of the blobs A and E_0 and blobequality of B and E_1 or by blobinequality of those pairs. The prover decides whether he proves blobequality twice or blobinequality twice and he does not say in advance what his choice is (and of course he has no choice otherwise it means that he can open blobs in two ways which violates our cryptographic assumption).

An even faster protocol is possible in case K^2 can be written as $f(d)$ and d is easy to find by the prover. Then the exclusive-or sum of the bits a an b is encoded by the product $A\ B$ assuming a and b are encoded by the blobs A and B. This is usual the case where G and H are subgroups of the multiplicative group of residues modulo a composite N and f is taking squares.

A final remark need to be made about the complexity to deal with the NAND gate. The equivalence between the five blobs D_i and the single blob C can be checked with five applications of the homomorphism f and six (even with five) multiplications in H and the same holds for the equivalence between the five blobs D_i and the two blobs A and B. Assuming the prover has already created blobs A, B and C the prover can create new blobs D_i and prepare possible answers on queries without having to do divisions in group G or H.

A Particular Example for a Blob

In [BC86] and [BCC] several implementations of blobs are given. We will give a slightly different one. This one is based on multiplications modulo a composite number N whose factorization has special properties (those numbers are often called Blum integers, the first reference for such composites is [Wil80]):

the number N is made by the verifier V.
the group G is the set of residues with positive Jacobi-symbol,
H is the set of squares modulo N,
K is an element of which the prover only knows a square root with negative Jacobi-symbol (and for which the verifier has proved with zero-knowledge techniques that he also knows a square root with positive Jacobi-symbol)

Multiparty Computations

Our idea can be exploited for multi party computations. The setting is as is known a number of participants each which secret bits and some say one bit function they want to compute. The model we have is using one outsider which does not collude with any of the other parties. The outsider makes a product of two primes, publishes this composite number plus one non-quadratic residue $K$with positive Jacobi-symbol. Then the outsider proves with an interactive zero-knowledge protocol the involved properties. Then

everybody encodes their bits using QRA [GM] with the modulus and residue K of the outsider, they pool their blobs, and they all go as one entity in interaction with the outsider. They produce sequences of five blobs, multiply each blob with a random square, apply a random cyclic permutation and a random decimation (in two of the four cases resulting in inverting the encoded bit), and offer the resulting sequence of five blobs to the outsider. The outsider responds with a single blob encoding the same bit as the five blobs. Using those answers the group can compute the value they want by sending the final sequence of five blobs (which encodes the final output or the inverse (the group knows but not the outsider)) and ask to open this blob. If this protocol is well designed the outsider will learn nothing and the participants only the final value. One can put the outsider to the test by asking him to prove that all his reductions from five blobs to one were good and such a protocol is basicly the same as our main protocol. The outsider could be a computer program which data can be erased by the group after the protocol. The group members can only find out secrets of each other when they violate QRA.

In principle our technique can be used to compute with blobs. The blob we get is then a sequence, having a length that is a five power, of single blobs. Two sets of five blobs (containing two commitments for *1* and three commitments for *0*) encodes a different bit if after a decimation with a factor two and a cyclic shift of one of the sets the corresponding blobs can shown to encode the same bit. And blobequality is done likewise but without the decimation. To change a blob (of length 5^{L+1})into a blob which encodes the inverted bit **(blob inversion)** we just rearrange all blobs of length 5^L by decimation with a factor two. Only blob inversion for the single blob might not be possible in general for the opponent but in the restricted general case we adopted for blobs it just the quotient of K with the blob. For the full general case the creator of the blob might have to create blobs to encode the inverted input values as well. Furthermore in order to compute with those sequences of blobs for each gate both sequences have to be made of the same length. This is done by blowing up the smallest of them with enough powers of five. To increase the length with a factor of five one replaces all "atom blobs" A by (A, B, O, B, A) where B is a blob encoding a different bit then A does and where O encodes the zero bit. Then any body can compute with those blobs to arrive at a very large blob.

Open Problems and Conclusion

It remains open how the ideas presented above can be used to conduct a general multiparty computation protocol with unconditional secrecy.
It may be concluded that in principle computing with blobs is possible (an open question in [BC86]) and now the question is can it also be done without increasing the length.

References

[BCC] Brassard, G., Chaum, D. and Crepeau, C. " Minimal disclosure proofs of knowledge" . Journal of computer and system sciences, **37**, 2 , october 1988, 156-189.

[BC86] Brassard, G and Crepeau, C. "Zero-Knowledge Simulation of Boolean circuits." Advances in Cryptology - Crypto '86, A.M. Odlyzko, Lecture Notes in Computer Science 263, 223-233, Springer-Verlag.

[GM] Goldwasser, S., and Micali, S., "Probabilistic encryption", JCSS, **28**, 2, 1984, 270-299.

[GMW86] Goldreich, O., Micali, S., Wigderson, A. "How to prove all NP statements in Zero-knowledge" Advances in Cryptology - Crypto '86, A.M. Odlyzko, Lecture Notes in Computer Science 263, 171-185, Springer-Verlag.

[GMS] Goldreich, O., Mansour, Y., Sipser, M. "Interactive Proof systems: Provers that never fail and random selection". Symp. on Found. of Comp. Sc., **28**, oct 87, 449-461.

[Wil80] Williams, H.C., "A modification of the RSA public-key encryption procedure." IEEE Trans. Inform. Theory, **26** (6), 726-729, November 1980.