Abstract. While it is fairly obvious that a secure (bit) commitment between two parties is impossible without a one-way function, we show that it is possible if the number of parties is at least 3. Then we show how our unconditionally secure (bit) commitment scheme for 3 parties can be used to arrange an unconditionally secure (bit) commitment between just two parties if they use a “dummy” (e.g., a computer) as the third party. We explain how our concept of a “dummy” is different from a well-known concept of a “trusted third party”. Based on a similar idea, we also offer an unconditionally secure k-n oblivious transfer protocol between two parties who use a “dummy”.

Finally, we suggest a protocol, without using a one-way function, for the so-called “mental poker”, i.e., a fair card dealing (and playing) over distance. Computational cost of our protocols is negligible to the point that all of them can be easily executed without a computer.

1. Introduction

In cryptography, a commitment scheme allows one to commit to a value while keeping it hidden, with the ability to reveal the committed value later. Commitments are used to bind a party to a value so that they cannot adapt to other messages in order to gain some kind of inappropriate advantage. They are important to a variety of cryptographic protocols including secure coin flipping, zero-knowledge proofs, and secure multi-party computation. See [4] or [6] for a general overview.

It is not hard to convince yourself that a secure (bit) commitment between two parties is impossible without some kind of encryption, i.e., without a one-way function. However, if the number of parties is at least 3, this becomes possible, as long as parties do not form coalitions to trick other party (or parties). It has to be pointed out though that formal definitions of commitment schemes vary strongly in notation and in flavor, so we have to be specific about our model. We give more formal details in Section 2, while here we just say, informally, that what we achieve is the following: if the committed values are just bits, then after the commitment stage of our scheme is completed, none of the parties can guess any other party’s bit with probability greater than $\frac{1}{2}$. We require in our scheme that there are $k$ secure channels for communication.

Research of the first author was partially supported by the Federal Agency of the Science and Innovations of Russia, State Contract No. 02.740.11.5192.

Research of the second author was partially supported by the NSF grant DMS-0914778.

1
between the parties, arranged in a circuit. We also show that less than $k$ secure channels is not enough.

Then, in Section 3, we show how our unconditionally secure (bit) commitment scheme for 3 parties can be used to arrange an unconditionally secure (bit) commitment between just two parties if they use a “dummy” (e.g., a computer) as the third party. We explain how our concept of a “dummy” is different from a well-known concept of a “trusted third party” and also from Rivest’s idea of a “trusted initializer” [8]. In particular, an important difference is that our “dummy” is not supposed to generate randomness. Based on a similar idea, we also offer, in Section 4, an unconditionally secure $k$-n oblivious transfer protocol between two parties who use a “dummy”.

Finally, in Section 5, we consider a related cryptographic primitive known as “mental poker”, i.e., a fair card dealing (and playing) over distance. Several protocols for doing this, most of them using encryption, have been suggested, the first by Shamir, Rivest, and Adleman [9], and subsequent proposals include [3] and [5]. As with the bit commitment, it is rather obvious that a fair card dealing between just two players over distance is impossible without a one-way function, or even a one-way function with trapdoor. However, it turns out to be possible if the number of players is $k \geq 3$. What we require though is that there are $k$ secure channels for communication between players, arranged in a circuit. We also show that our protocol can, in fact, be adapted to deal cards to just 2 players. Namely, if we have 2 players, they can use a “dummy” player (e.g. a computer), deal cards to 3 players, and then just ignore the “dummy”’s cards, i.e., “put his cards back in the deck”. An assumption on the “dummy” player is that he cannot generate any randomness, so randomness has to be supplied to him by the two “real” players. Another assumption is that there are secure channels for communication between either “real” player and the “dummy”. We believe that this model is adequate for 2 players who want to play online but do not trust the server. “Not trusting” the server exactly means not trusting with generating randomness. Other, deterministic, operations can be verified at the end of the game; we give more details in Section 5.2.

We note that the only known (to us) proposal for dealing cards to $k \geq 3$ players over distance without using one-way functions was published in [1], but their protocol lacks the simplicity, efficiency, and some of the functionalities of our proposal; this is discussed in more detail in our Section 6. Here we just mention that computational cost of our protocols is negligible to the point that they can be easily executed without a computer.

2. (Bit) commitment

While it is fairly obvious that a secure (bit) commitment between two parties is impossible without a one-way function, we show here that it is possible if the number of parties is at least 3. Generalizing the standard concept (see e.g. [4]) of a two-party (bit) commitment scheme, we define an $n$-party (bit) commitment scheme to be a two-phase protocol through which each of the $n$ parties can commit himself to a value such that the following two requirements are satisfied:
Each participant \( PP_i \) (\( \mathbb{P} \)) integers \( n \) the parties exchange various pieces of information about their integers one. Suppose they want to commit to integers ambiguity. each party’s committed value can be recovered (collectively by other parties) without later the parties perform the decommitment phase (sometimes called the reveal phase), each party’s committed value can be recovered (collectively by other parties) without ambiguity.

To make our ideas more transparent, we start with the simplest case where there are just 3 parties: \( P_1, P_2, \) and \( P_3, \) and no two of them form a coalition against the third one. Suppose they want to commit to integers \( n_1, n_2, \) and \( n_3 \) (modulo some \( m \geq 2 \)), respectively. More precisely, the scenario is as follows. During the commitment phase, the parties exchange various pieces of information about their integers \( n_i \). After that, the parties “decommit”, or reveal, their integers and prove to each other that the integers \( n_i \) that they revealed are the same that they committed to.

All computations below are performed modulo a fixed integer \( m \geq 2 \).

1. Each participant \( P_i \) randomly splits his integer \( n_i \) in a sum of two integers: \( n_i = r_i + s_i \). If the participants want to commit to bits rather than integers, then \( P_i \) would split the “0” bit as either 0+0 or 1+1, and the “1” bit as either 0+1 or 1+0.

2. (**Commitment phase.**) \( P_1 \) sends \( r_1 \) to \( P_2 \), then \( P_2 \) sends \( r_1 + r_2 \) to \( P_3 \), then \( P_3 \) sends \( r_1 + r_2 + r_3 \) to \( P_1 \). In the “opposite direction”, \( P_3 \) sends \( s_3 \) to \( P_2 \), then \( P_2 \) sends \( s_2 + s_3 \) to \( P_1 \), then \( P_1 \) sends \( s_1 + s_2 + s_3 \) to \( P_3 \).

   After the commitment phase, \( P_1 \) has \( s_1, s_2 + s_3, r_1, \) and \( r_1 + r_2 + r_3 \) (therefore also \( r_2 + r_3 \)), so he cannot possibly recover any \( n_i \) other than his own. (He can recover \( n_2 + n_3 \), but this does not give him any information about either \( n_2 \) or \( n_3 \)). Then, \( P_2 \) has \( s_2, s_3, r_1, \) and \( r_2 \), so he, too, cannot possibly recover any \( n_i \) other than his own. Finally, \( P_3 \) has \( s_3, r_3, r_1 + r_2, \) and \( s_1 + s_2 + s_3 \) (therefore also \( s_2 + s_3 \)), so he, too, cannot possibly recover any \( n_i \) other than his own. (He can recover \( n_1 + n_2 \), but this does not give him any information about either \( n_1 \) or \( n_2 \)).

3. (**Decommitment phase starts.**) Note that during the decommitment steps below, each participant transmits information that somebody else had committed to before. This way, each piece of transmitted information can be corroborated by two parties, which prevents cheating since we are assuming that no two participants form a coalition.

4. \( P_3 \) sends \( n_1 + n_2 \) to both \( P_1 \) and \( P_2 \). Now \( P_1 \) knows \( n_2 \), and \( P_2 \) knows \( n_1 \).

5. \( P_2 \) sends \( r_1 \) to \( P_3 \). Now \( P_3 \) can recover \( r_2 \) from \( r_1 \) and \( r_1 + r_2 \).

6. \( P_1 \) sends \( s_2 + s_3 \) to \( P_3 \). Now \( P_3 \) can extract \( s_2 \) from this sum, and then, since he has \( r_2 \), recover \( n_2 \), and then also \( n_1 \) since \( P_3 \) already knows \( n_1 + n_2 \).

7. \( P_1 \) sends \( r_1 + r_2 + r_3 \) to \( P_2 \). Now \( P_2 \) can recover \( r_3 \) and therefore \( n_3 = r_3 + s_3 \).

This protocol can be obviously generalized to \( 3m \) participants for arbitrary \( m \geq 1 \) by splitting the players into triples and applying the above protocol to each triple.
can also be generalized to an arbitrary number \( k \geq 3 \) of participants with a circular arrangement of \( k \) secure channels, but we leave details to the reader.

**Remark 1.** A question that one might now ask, if only out of curiosity, is: would this scheme work with any arrangement of secure channels other than a union of disjoint circuits of length \( \geq 3 \)? The answer to this question is “no”. Indeed, if in the graph of secure channels there is a vertex (corresponding to \( P_1 \), say) of degree 0, then any information sent out by \( P_1 \) will be available to everybody, so other participants will know \( n_1 \) unless \( P_1 \) uses a one-way function to conceal it. If there is a vertex (again, corresponding to \( P_1 \)) of degree 1, this would mean that \( P_1 \) has a secure channel of communication with just one other participant, say \( P_2 \). Then any information sent out by \( P_1 \) will be available at least to \( P_2 \), so \( P_2 \) will know \( n_1 \) unless \( P_1 \) uses a one-way function to conceal it. So, every vertex in the graph should have degree at least 2, which implies that every vertex is included in a circuit. It follows, in particular, that the total number of secure channels should be at least \( k \), by the number of participants.

3. (Bit) commitment between two parties

Now we show how our unconditionally secure commitment scheme for 3 parties from Section 2 can be used to arrange an unconditionally secure commitment between just two parties. This is similar, in spirit, to the idea of Rivest [8], where an extra participant is introduced to bring the number of parties up to 3. However, an important difference between our proposal and that of [8] is that the extra participant in [8] is a “trusted initializer”, which means that (i) he is allowed to generate randomness; (ii) he can transmit information to “real” participants over secure channels.

By contrast, our extra participant is a “dummy”, i.e., (i) he is not allowed to generate randomness; (ii) he can receive information from “real” participants over secure channels and perform simple arithmetic operations.

One possible real-life interpretation of such a “dummy” would be an online calculator that can combine inputs from different users. Also note that in our scheme below the “dummy” is unaware of the committed values, which is useful in case the two “real” parties do not want their commitments to ever be revealed to the third party; for example, such a “dummy” could be a mediator between two parties negotiating a divorce settlement.

Thus, let \( A \) (Alice) and \( B \) (Bob) be two “real” participants, and \( D \) (Dummy) the “dummy”. Suppose \( A \) and \( B \) want to commit to integers \( n_1 \) and \( n_2 \), respectively.

1. \( A \) and \( B \) randomly split their integers \( n_i \) in a sum of two integers: \( n_i = r_i + s_i \).
2. (Commitment.) \( A \) sends \( s_1 \) to \( B \), and \( B \) sends \( r_2 \) to \( A \). Then, \( A \) sends \( r_1 + r_2 \) to \( D \), and \( B \) sends \( s_1 + s_2 \) to \( D \).
3. (Decommitment.) \( D \) reveals \( r_1 + r_2 + s_1 + s_2 = n_1 + n_2 \) both to \( A \) and \( B \).
4. Now \( A \) knows \( (n_1 + n_2) - n_1 = n_2 \), and \( B \) knows \( (n_1 + n_2) - n_2 = n_1 \), so cheating by either party is impossible.
4. $k$-n OBVIOUS TRANSFER

An oblivious transfer protocol is a protocol by which a sender sends some information to the receiver, but remains oblivious as to what is received. The first form of oblivious transfer was introduced in 1981 by Rabin [7]. Rabin’s oblivious transfer was later shown to be equivalent to “1-2 oblivious transfer”; the latter was subsequently generalized to 1-n oblivious transfer and to $k$-n oblivious transfer [2]. In the latter case, the receiver obtains a set of $k$ messages from a collection of $n$ messages. The set of $k$ messages may be received simultaneously (“non-adaptively”), or they may be requested consecutively, with each request based on previous messages received. All the aforementioned constructions use encryption, so in particular they use one-way functions. The first proposal that did not use one-way functions (and therefore offered unconditionally secure oblivious transfer) appeared in the paper by Rivest [8] that we have already cited in our Section 3.

In this section, we offer an unconditionally secure $k$-n oblivious transfer protocol that is essentially different from that of Rivest in a similar way that our bit commitment protocol in Section 3 is different from Rivest’s unconditionally secure bit commitment protocol [8]. More specifically, the extra participant in [8] is a “trusted initializer”, which means, in particular, that (i) he is allowed to generate randomness; (ii) he can “consciously” transmit information to “real” participants over secure channels.

By contrast, our extra participant is a “dummy”, i.e., (i) he is not allowed to generate randomness; (ii) he can receive information from “real” participants over secure channels, but he transmits information upon specific requests only.

Again, let A (Alice) and B (Bob) be two “real” participants, and D (Dummy) the “dummy”, e.g., a computer. Suppose A has a collection of $n$ messages, and B wants to obtain $k$ of these messages, without A knowing which messages B has received. Suppose that all messages are integers $m_i$, $1 \leq i \leq n$.

1. A randomly splits her integers $m_i$ in a sum of two integers: $m_i = r_i + s_i$.
2. A sends the (ordered) set of all $r_i$, $1 \leq i \leq n$, to D, and the (ordered) set of all $s_i$, $1 \leq i \leq n$, to B.
3. B sends to D the set of indices $j_1, \ldots, j_k$ corresponding to the messages $m_j$ he wants to receive.
4. D sends to B the (ordered) set $r_{j_1}, \ldots, r_{j_k}$.
5. B recovers $m_{j_1}, \ldots, m_{j_k}$ as a sum of relevant $r_j$ and $s_j$.

5. MENTAL POKER

“Mental poker” is the common name for a set of cryptographic problems that concerns playing a fair game over distance without the need for a trusted third party. One of the ways to describe the problem is: how can 2 players deal cards fairly over the phone? Several protocols for doing this have been suggested, including [9], [3], [5] and [1]. As with the bit commitment, it is rather obvious that a fair card dealing to two players over distance is impossible without a one-way function, or even a one-way function with trapdoor. However, it turns out to be possible if the number of players is at least 3, assuming, of course, that there are secure channels for communication
between at least some of the players. In our proposal, we will be using \( k \) secure channels for \( k \geq 3 \) players \( P_1, \ldots, P_k \), and these \( k \) channels will be arranged in a circuit: \( P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_k \rightarrow P_1 \).

To begin with, suppose there are 3 players: \( P_1, P_2, \) and \( P_3 \) and 3 secure channels: \( P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_1 \).

The first protocol, Protocol 1 below, is for distributing all integers from 1 to \( m \) to the players in such a way that each player gets about the same number of integers. (For example, if the deck that we want to deal has 52 cards, then two players should get 17 integers each, and one player should get 18 integers.) In other words, Protocol 1 allows one to randomly split a set of \( m \) integers into 3 disjoint sets.

The second protocol, Protocol 2, is for collectively generating random integers modulo a given integer \( M \). This very simple but useful primitive can be used: (i) for collectively generating, uniformly randomly, a permutation from the group \( S_m \). This will allow us to assign cards from a deck of \( m \) cards to the \( m \) integers distributed by Protocol 1; (ii) introducing “dummy” players as well as for “playing” after dealing cards.

5.1. **Protocol 1.** For notational convenience, we are assuming below that we have to distribute integers from 1 to \( r = 3s \) to 3 players.

To begin with, all players agree on a parameter \( N \), which is a positive integer of a reasonable magnitude, say, 10.

1. Each player \( P_i \) picks, uniformly randomly, an integer (a “counter”) \( c_i \) between 1 and \( N \), and keeps it private.
2. \( P_1 \) starts with the “extra” integer 0 and sends it to \( P_2 \).
3. \( P_2 \) sends to \( P_3 \) either the integer \( m \) he got from \( P_1 \), or \( m + 1 \). More specifically, if \( P_2 \) gets from \( P_1 \) the same integer \( m \) less than or equal to \( c_2 \) times, then he sends \( m \) to \( P_3 \); otherwise, he sends \( m + 1 \) and keeps \( m \) (i.e., in the latter case \( m \) becomes one of “his” integers). Having sent out \( m + 1 \), he “resets his counter”, i.e., selects, uniformly randomly between 1 and \( N \), a new \( c_2 \). He also resets his counter if he gets the number \( m \) for the first time, even if he does not keep it.
4. \( P_3 \) sends to \( P_1 \) either the integer \( m \) he got from \( P_2 \), or \( m + 1 \). More specifically, if \( P_3 \) gets from \( P_2 \) the same integer \( m \) less than or equal to \( c_3 \) times, then he sends \( m \) to \( P_1 \); otherwise, he sends \( m + 1 \) and keeps \( m \). Having sent out \( m + 1 \), he selects a new counter \( c_3 \). He also resets his counter if he gets the number \( m \) for the first time, even if he does not keep it.
5. \( P_1 \) sends to \( P_2 \) either the integer \( m \) he got from \( P_3 \), or \( m + 1 \). More specifically, if \( P_1 \) gets from \( P_3 \) the same integer \( m \) less than or equal to \( c_1 \) times, then he sends \( m \) to \( P_2 \); otherwise, he sends \( m + 1 \) and keeps \( m \). Having sent out \( m + 1 \), he selects a new counter \( c_1 \). He also resets his counter if he gets the number \( m \) for the first time, even if he does not keep it.
6. This procedure continues until one of the players gets \( s \) integers (not counting the “extra” integer 0). After that, a player who already has \( s \) integers just “passes along” any integer that comes his way, while other players keep following the above procedure until they, too, get \( s \) integers.
The protocol ends as follows. When all 3s integers, between 1 and 3s, are distributed, the player who got the last integer, 3s, keeps this fact to himself and passes this integer along as if he did not “take” it.

The process ends when the integer 3s makes $N + 1$ “full circles”.

We note that the role of the “extra” integer 0 is to prevent $P_3$ from knowing that $P_2$ has got the integer 1 if it happens so that $c_2 = 1$ in the beginning.

We also note that this protocol can be generalized to arbitrarily many players in the obvious way, if there are $k$ secure channels for communication between $k$ players, arranged in a circuit.

5.2. Protocol 2. Now we describe a protocol for generating random integers modulo some integer $M$ collectively by 3 players. As in Protocol 1, we are assuming that there are secure channels for communication between the players, arranged in a circuit.

1. $P_2$ and $P_3$ uniformly randomly and independently select private integers $n_2$ and $n_3$ (respectively) modulo $M$.
2. $P_2$ sends $n_2$ to $P_1$, and $P_3$ sends $n_3$ to $P_1$.
3. $P_1$ computes the sum $m = n_2 + n_3$ modulo $M$.

Note that neither $P_2$ nor $P_3$ can cheat by trying to make a “clever” selection of their $n_i$ because the sum, modulo $M$, of any integer with an integer uniformly distributed between 0 and $M - 1$, is an integer uniformly distributed between 0 and $M - 1$.

Finally, $P_1$ cannot cheat simply because he does not really get a chance: if he miscalculates $n_2 + n_3$ modulo $M$, this will be revealed at the end of the game. (All players keep contemporaneous records of all transactions, so that at the end of the game, correctness could be verified.)

To generalize Protocol 2 to arbitrarily many players $P_1, \ldots, P_k$, $k \geq 3$, we can just engage 3 players at a time in running the above protocol. If, at the same time, we want to keep the same circular arrangement of secure channels between the players that we had in Protocol 1, i.e., $P_1 \rightarrow P_2 \rightarrow \ldots P_k \rightarrow P_1$, then 3 players would have to be $P_{i+1}$, $P_i$, $P_{i+2}$, where $i$ would run from 1 to $k$, and the indices are considered modulo $k$.

Protocol 2 can now be used to collectively generate, uniformly randomly, a permutation from the group $S_m$. This will allow us to assign cards from a deck of $m$ cards to the $m$ integers distributed by Protocol 1. Generating a random permutation from $S_m$ can be done by taking a random integer between 1 and $m$ (using Protocol 2) sequentially, ensuring that there is no repetition. This “brute-force” method will require occasional retries whenever the random integer picked is a repeat of an integer already selected. A simple algorithm to generate a permutation from $S_m$ uniformly randomly without retries, known as the Knuth shuffle, is to start with the identity permutation or any other permutation, and then go through the positions 1 through $(m - 1)$, and for each position $i$ swap the element currently there with an arbitrarily chosen element from positions $i$ through $m$, inclusive (again, Protocol 2 can be used here to produce a random integer between $i$ and $m$). It is easy to verify that any permutation of $m$ elements will be produced by this algorithm with probability exactly $\frac{1}{m!}$, thus yielding a uniform distribution over all such permutations.
After this is done, we have \( m \) cards distributed uniformly randomly to the players, i.e., we have:

**Proposition 1.** If \( m \) cards are distributed to \( k \) players using Protocols 1 and 2, then the probability for any particular card to be distributed to any particular player is \( \frac{1}{k} \).

### 5.3. Using “dummy” players while dealing cards

We now show how a combination of Protocol 1 and Protocol 2 can be used to deal cards to just 2 players. If we have 2 players, they can use a “dummy” player (e.g. a computer), deal cards to 3 players as in Protocol 1, and then just ignore the “dummy”’s cards, i.e., “put his cards back in the deck”. We note that the “dummy” in this scenario would not generate randomness; it will be generated for him by the other two players using Protocol 2. Namely, if we call the “dummy” \( P_3 \), then the player \( P_1 \) would randomly generate \( c_{31} \) between 1 and \( N \) and send it to \( P_3 \), and \( P_2 \) would randomly generate \( c_{32} \) between 1 and \( N \) and send it to \( P_3 \). Then \( P_3 \) would compute his random number as \( c_3 = c_{31} + c_{32} \mod N \).

Similarly, “dummy” players can help \( k \) “real” players each get a fixed number \( s \) of cards, because Protocol 1 alone is only good for distributing all cards in the deck to the players, dealing each player about the same number of cards. We can introduce \( m \) “dummy” players so that \((m + k) \cdot s\) is approximately equal to the number of cards in the deck, and position all the “dummy” players one after another as part of a circuit \( P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_{m+k} \rightarrow P_1 \). Then we use Protocol 1 to distribute all cards in the deck to \((m + k)\) players taking care that each “real” player gets exactly \( s \) cards. As in the previous paragraph, “dummy” players have “real” ones generate randomness for them using Protocol 2.

### 6. Summary of the properties of our card dealing (Protocols 1 and 2)

Here we summarize the properties of our Protocols 1 and 2 and compare, where appropriate, our protocols to the card dealing protocol of [1].

1. **Uniqueness of cards.** Yes, by the very design of Protocol 1.

2. **Uniform random distribution of cards.** Yes, because of Protocol 2; see our Proposition 1 in Section 5.2.

3. **Complete confidentiality of cards.** Yes, by the design of Protocol 1.

4. **Number of secure channels for communication between \( k \geq 3 \) players:** \( k \), arranged in a circuit.

   By comparison, the card dealing protocol of [1] requires \( 3k \) secure channels.

5. **Average number of transmissions between \( k \geq 3 \) players:** \( O(N^2 mk) \), where \( m \) is the number of cards in the deck, and \( N \approx 10 \). This is because in Protocol 1,
the number of circles (complete or incomplete) each integer makes is either 1 or the minimum of all the counters $c_i$ at the moment when this integer completes the first circle. Since the average of $c_i$ is at most $\frac{N}{2}$, we get the result because within one circle (complete or incomplete) there are at most $k$ transmissions. We note that in fact, there is a precise formula for the average of the minimum of $c_i$ in this situation: $\frac{\sum_{j=1}^{N} \frac{j^k}{N^k}}{N}$, which is less than $\frac{N}{2}$ if $k \geq 2$.

By comparison, in the protocol of [1] there are $O(mk^2)$ transmissions.

6. **Total length of transmissions between $k \geq 3$ players**: $\frac{N}{2} mk \cdot \log_2 m$ bits. This is just the average number of transmissions times the length of a single transmission, which is a positive integer between 1 and $m$.

By comparison, total length of transmissions in [1] is $O(mk^2 \log k)$.

7. **Computational cost of Protocol 1**: 0 (because there are no computations, only transmissions).

By comparison, the protocol of [1] requires computing products of up to $k$ permutations from the group $S_k$ to deal just one card; the total computational cost therefore is $O(mk^2 \log k)$.

Acknowledgement. Both authors are grateful to Max Planck Institut für Mathematik, Bonn for its hospitality during the work on this paper.

References
