Dealing Cards in Poker Games

Philippe Golle
Palo Alto Research Center
pgolle@parc.com

***Abstract*— This paper proposes a new protocol for shuffling and dealing cards, that is designed specifically for games of mental poker. Our protocol takes advantage of two features of poker games that are overlooked by generic card-shuffling protocols: 1) cards in poker games are dealt in rounds, with betting in-between, rather than all at once and 2) the total number of cards dealt in a game of poker is small (it depends on the number of players but is typically less than half the deck). With these observations in mind, we propose a protocol that spreads the computational cost of dealing cards more evenly across rounds. Compared to protocols that shuffle the whole deck upfront, our approach offers a dramatic decrease in latency and overall computational cost. Our protocol is fair, private and robust. It is ideally suited for resource-constrained devices such as PDAs.**

***Index Terms*— Shuffle, Mental Poker, Mix, Latency, ElGamal.**

I. Introduction

Mental, or electronic, card games require protocols for shuffling a deck of cards and for dealing cards to players in a way that is private: players must learn no information about cards dealt face down to other players. In the abstract, this problem is well studied. Various protocols [16], [12] let a number of players jointly compute a permutation of n elements, in a way that is private and very efficient asymptotically. These generic protocols can naturally be used to shuffle a deck of 52 cards, and deal, for example, a game of poker. However, generic protocols are not designed specifically for real card games, and thus offer far from optimal computational efficiency and latency in actual games.

This paper proposes a new protocol for shuffling and dealing cards, that is designed specifically for games of mental poker. Our protocol takes advantage of two features of poker games that are overlooked by generic card-shuffling protocols: 1) cards in poker games are dealt in rounds, with betting in-between, rather than all at once and 2) the total number of cards dealt in a game of poker is small (it depends on the number of players but is typically less than half the deck). With these observations in mind, we propose a protocol that spreads the computational cost of shuffling and dealing cards more evenly across rounds. Compared to protocols that shuffle the whole deck upfront, our approach offers a dramatic decrease in latency and overall computational cost.

For example, the total computational cost of shuffling and dealing cards for one game of "Texas Hold'em" among 5 players is 66% lower with our protocol than with the most efficient generic protocol [12]. More importantly, the initial setup latency (before the first round of betting) is 85% lower. Our protocol offers similar improvements for "Seven Card Stud" and other popular poker games [1].

Our approach to dealing cards is conceptually very simple. Unlike generic protocols, we do not shuffle the whole deck of cards in a time-consuming initial step. Rather, we generate cards one by one when they are needed. Let the interval $[1, \ldots, 52]$ represent the 52 cards in a deck. To generate a card, the players jointly compute the encryption $E(c)$ of a random number $c \in [1, \ldots, 52]$. The players then compare the ciphertext $E(c)$ with all the (encrypted) cards $E(c_1), \ldots, E(c_t)$ that have already been dealt in the current game. If a match is found, the newly generated card $E(c)$ is discarded and the players try again generating a card. If no match is found, the card $E(c)$ is a valid new card. To deal face up, the players jointly decrypt the card and output c. To deal face down, the players help the recipient of the card decrypt, in such a way that only the recipient learns c.

This protocol is extremely efficient when the total number of cards dealt in a game is small compared to the number of cards in the deck (nearly all poker games have that property). Is is, however, unsuitable for dealing a *whole* deck of cards. Furthermore, the computational cost of dealing cards is not all incurred upfront, but rather it is spread across multiple rounds of betting, thus offering players much reduced upfront latency. As will be clear in the rest of this paper, the protocol is private. No player or coalition of players (below a threshold) can influence the card generation protocol, nor learn anything about any of the cards dealt face down to other players. Our protocol is ideally suited for resource-constrained devices such as PDAs or even next-generation cell phones.

Overview. In the rest of this section, we introduce our model and survey briefly known approaches to dealing cards. In Section II, we give a high-level description of our protocol. We discuss specific implementations of our protocol with an RSA-based encryption scheme (Section III) and with ElGamal (Section IV). In Section V, we discuss the efficiency of our protocol for specific games and offer detailed evidence that it is superior to generic mixing protocols.

A. Model

Games of mental poker require the ability to shuffle a deck of cards, and to deal cards either face up or face down (other requirements of online poker, such as the ability to transfer payments between players, are outside the scope of this paper). Note that the shuffling operation may be implicit: in our protocol, shuffling and dealing occur concurrently. The properties desired of mental poker with respect to shuffling and dealing cards are exactly derived from the properties of "real" poker games. Specifically, a player or coalition of

players should not be able to influence shuffling in any way, nor learn anything about a card dealt face down to anyone outside the coalition.

Adversarial model. Let k be the number of players. We assume that the adversary can actively corrupt up to a minority of players, and passively corrupt up to $k-1$ players. Note that this is the strongest possible adversarial model. We desire the following properties of our protocol:

- **Correctness.** Every card dealt, whether face up or face down, whether to the adversary or not, should be drawn uniformly at random from the set of cards left in the deck.
- **Privacy.** The adversary learns no information about cards dealt face down to players that the adversary does not control.
- **Robustness.** We need not worry about one player dropping out (after all, a player is allowed to fold), but we must prevent one player from preventing the others from continuing a round.

Poker is usually played with a standard 4-suit 52-card deck, but a joker or other wild cards may be added. In what follows, we assume that a deck of cards consists of exactly 52 cards, but it will be clear that our protocols can easily be modified to accommodate larger (or smaller) decks. We represent the cards in the deck with the set of integers modulo 52, i.e. the set $\{0, \ldots, 51\}$.

Efficiency. There are many ways to measure the efficiency of a protocol: round complexity, communication cost, computational cost, etc. For card dealing protocols, computational costs dwarf all other costs. This paper thus focuses exclusively on computational cost, which we measure as the number of modular exponentiations that each player must perform.

B. Related Work

In this section, we review briefly generic approaches to shuffling a deck of cards.

Trusted Third Party. A Trusted Third Party (TTP) is a party trusted by all players to mix and deal cards fairly. For example, an online casino acts as a TTP for the players that are registered with it. A TTP is a simple and efficient solution to dealing cards, but it constitutes a single point of failure and offers very weak security. For example, a malicious TTP may collude with some players against others. Even an honest TTP may not mix properly (see [2] for an example of an off-by-one error in a shuffling algorithm). These weaknesses disqualify TTP-based approaches from serious consideration.

Distributing trust. The first scheme to successfully distribute trust among all players is due to Crépeau [8]. As long as a majority of players are honest, all players are assured that cards are dealt fairly and privately. The computational cost of this scheme is unfortunately prohibitively high. In 1994, an implementation of [8] on three Sparc workstations is reported to have taken eight hours to shuffle a deck of cards [9].

Mix network. Efficient schemes [14], [3] for dealing cards without a TTP all rely on a primitive known as a mix server. A mix server [6], [16] takes a set of input ciphertexts and outputs equivalent ciphertexts (i.e. ciphertexts that decrypt to the same plaintexts), in a randomly permuted order. The permutation that matches input to output ciphertexts is known only to the mix server. A verifiable mix server also outputs a proof of correct mixing that allows a verifier to check that the mixing operation was done correctly.

To shuffle cards, the players first jointly encrypt the deck of cards. Each player then acts as a mix server and mixes the cards according to a secret permutation. The result of this operation is a shuffled deck of cards, from which cards can be dealt one by one.

To mix a deck of 52 cards, the most efficient verifiable mix network is a construction called Millimix [12] that relies on a permutation network. To mix 52 cards, we need $d = 321$ comparitors. The following table shows the real cost per player (for a total of k players) of mixing 52 cards with Millimix. The cost is measured in terms of the number of modular exponentiations.

Operation	Cost	$k = 5$
Deck setup (total)	(3k+4)d	**6099**
Mixing and proof	$7d$	2247
Verifying others' proofs	$3d(k-1)$	3852
Dealing a card (total)	$4 + 2k$	**14**
Decryption and proof	6	6
Verifying others' proofs	$2(k-1)$	8

As this table illustrates, the main drawback of mix networks is that the whole computational cost of shuffling is borne upfront, resulting in very high latency before a game begins.

II. Protocol Overview

In this section, we give an overview of our protocol. To deal a card, the players jointly generate a semantically secure encryption $E(c)$ of a random integer $c \in \{0, \ldots, 51\}$ without revealing c. To avoid dealing the same card twice, the players must ensure that the newly generated card $E(c)$ is different from all the cards already dealt. The difficulty is that this comparison must be made between encrypted cards, in a way that reveals only whether $E(c)$ has already been dealt. If $E(c)$ was already dealt, the players repeat the protocol and try a new card $E(c')$ until one is found that has not already been dealt.

A complete description of the protocol follows. The protocol requires a semantically secure public-key encryption scheme E with the following properties:

- The key generation and decryption algorithms for E can be distributed among k players.
- E is additively homomorphic, i.e. $E(m_1)E(m_2) = E(m_1 + m_2)$.
- Given two ciphertexts $E(c)$ and $E(c')$, there is a protocol that allows the joint holders of the private key to learn whether $c = c' \mod 52$ without revealing anything else.

We discuss encryption schemes with these properties in Sections III and IV. Given an encryption scheme E with the properties above, the protocol for dealing cards is as follows.

Group establishment. The players jointly generate and share the public and private parameters of the encryption scheme E. Every player receives the public parameters and a share of the private key. After a group is established, the same public and private parameters can be reused to deal multiple decks of cards. The group establishment protocol only needs to be run again if a player leaves, a new player joins, or a new group is formed.

Deck setup. The players keep a list $L = \{E(c_1), \ldots, E(c_t)\}$ that consists of encryptions of all the cards that have been dealt from the current deck. The list L includes both the cards dealt face up and the cards dealt face down. When a new deck is setup, we initialize $L = \emptyset$.

Dealing a card. The protocol to deal a card, whether face up or face down, is as follows:

1) Every player P_i chooses $r_i \stackrel{\mathrm{R}}{\leftarrow} \{0, \ldots, 51\}$, computes the ciphertext $E(r_i)$ and outputs a non-malleable commitment to $E(r_i)$.
2) Each player P_i then reveals $E(r_i)$, and all players verify that all commitments are correct. If one or more commitments are incorrect, the protocol aborts and the honest players establish a new group that excludes the dishonest players.
3) Using the additive homomorphism of E, the players compute $E(c)$, where $c = \sum_i r_i$.
4) If $L \neq \emptyset$, the players must test whether the card $E(c)$ already belongs to L. More precisely, for every ciphertext $E(c') \in L$, the players run the joint protocol to test whether $c = c' \mod 52$. If there exists $E(c') \in L$ such that $c = c' \mod 52$, the players discard the card $E(c)$ and restart the card dealing protocol in step 1. Note that the card $E(c')$ that had already been dealt earlier is unaffected by the collision.
5) The players add $E(c)$ to the list L. To deal the card face up, the players jointly decrypt $E(c)$ and output $c \mod 52$. To deal the card face down to player P_j, all players other than P_j partially decrypt $E(c)$ under their share of the private key. The resulting ciphertext can be decrypted by P_j alone.

In the rest of this paper, we present two encryption schemes with the required properties, and examine for both the computational cost of establishing a group, setting up a deck and dealing a card.

Expected number of collisions to deal a cards. The expected total number of collisions to deal a cards is approximately $\frac{1}{52}\left(\frac{a(a-1)}{2}\right)$ as long as $a \ll 52$.

III. Dense Probabilistic Encryption

Dense probabilistic encryption is a semantically secure public-key encryption scheme proposed by Benaloh [4]. This encryption scheme satisfies 2 of the 3 properties we require: it is additively homomorphic and allows for modular plaintext comparison, but unfortunately it has no efficient algorithm for distributed key generation (as discussed below, we must therefore rely on a trusted third party for key generation).

Dense probabilistic encryption has an additive homomorphism modulo r, where r is an odd integer that parameterizes the encryption function. For our application, we set $r = 53$ and thus obtain an encryption scheme with an additive homomorphism modulo 53 (rather than 52). This discrepancy is easy to deal with: the players add a 53^{rd} special card to the standard 52-card deck. This special 53^{rd} card is added to the list L during deck setup to ensure it is never dealt.

We review briefly the Benaloh encryption scheme below (see [4] for details).

Key Generation. Let r be an odd integer (in our application $r = 53$). Let p, q be two large primes such that r divides $p-1$, $\gcd(r, (p-1)/r) = 1$ and $\gcd(r, q-1) = r$. Let $N = pq$. Let $y \in \mathbb{Z}_N^*$ such that $y^{(p-1)(q-1)/r} \neq 1 \mod N$. The public key consists of N and y, and the private key is $d = (p-1)(q-1)/r$.

Distributed Key Generation. Existing protocols [5], [11] for distributed RSA-key generation can be adapted to distributed key generation for dense probabilistic encryption, but these protocols are too inefficient for practical use. In practice, we must rely on a trusted third party to compute N and y as above, and distribute additive shares of $d = (p-1)(q-1)/r$ to the players. Note that this trusted third party plays a very limited role: its involvement is limited to key generation and it is never used again afterward. Such a limited third party that helps establish groups of players may be acceptable.

Encryption. To encrypt $m \in \mathbb{Z}_r$, choose $u \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_N^*$ and let $E(m) = y^m u^r$.

Decryption. Consider a ciphertext $z = E(m) = y^m u^r$. Recall that $d = (p-1)(q-1)/r$. We have $z^d = y^{md}$. We build a look-up table of the values y^d for $m \in \{0, \ldots, r-1\}$, and decrypt a ciphertext z by looking up the value z^d in the table.

Additive homomorphism modulo r. Let $m_1, m_2 \in \mathbb{Z}_r$, and let $z_1 = E(m_1)$ and $z_2 = E(m_2)$. It is easy to verify that $z_1 z_2 = E(m_1 + m_2)$.

Test of plaintext equality modulo r. Let $E(m_1)$ and $E(m_2)$ be two ciphertexts. The goal is to determine whether $m_1 = m_2 \mod r$ without revealing any other information. The players first compute $E(m_1)/E(m_2) = E(m)$, where $m = m_1 - m_2 \mod r$. The problem is to determine whether $m = 0 \mod r$. Each player P_i in turn chooses $\alpha_i \stackrel{\mathrm{R}}{\leftarrow} \{1, \ldots, 52\}$ and outputs $E(m)^{\alpha_i}$ together with a proof of correct exponentiation (essentially a non-interactive Schnorr signature [18]). Let $\alpha = \sum_i \alpha_i$. The players compute $\prod_i E(m)^{\alpha_i} = E(m)^{\alpha}$. Note that $E(m)^{\alpha} = y^{m\alpha} u^{r\alpha}$ and therefore $E(m)^{\alpha}$ is an encryption of $m\alpha \mod r$. Furthermore, since $r = 53$ is prime in our application, $m\alpha = 0 \mod r$ if and only if $m = 0 \mod r$. Otherwise, for $m \neq 0 \mod r$, the value $m\alpha$ is uniformly distributed over $\{1, \ldots, 52\}$. The players jointly decrypt $E(m)^{\alpha}$ and output $m_1 = m_2 \mod r$ if and only if $m\alpha = 0 \mod r$.

The computational cost of this protocol is given in the following table, expressed as the number of modular exponentiations per player:

Operation	Cost
Deck setup	2
Test of plaintext equality	$4k$
Dealing a card	$4k\|L\|/(1-\|L\|/52)$

Other encryption schemes offer semantically secure dense probabilistic encryption, such as a scheme by Naccache and Stern [15], but these schemes are also based on an RSA modulus for which there is no efficient distributed key generation algorithm.

IV. ElGamal Implementation

In this section, we propose an implementation of our protocol for dealing cards that is based on the ElGamal encryption scheme [10]. The ElGamal implementation is less efficient than the implementation described in the previous section, but it offers the significant advantage of efficient distributed key generation without relying on a trusted third party. Recall that we let k denote the number of players. We review first the definition and useful properties of ElGamal.

Key generation. Let p be a 1024-bit prime and q be a 160-bit prime such that $q|(p-1)$. Let $g \in \mathbb{Z}_p^*$ be an element of order q, and let G be the multiplicative subgroup of Z_p^* generated by g. The parameters p, q and g are public. Let $x \in \mathbb{Z}_q^*$ be a private key and $y = g^x$ the corresponding public key. ElGamal is semantically secure if the Decisional Diffie-Hellman (DDH) assumption holds in the group G.

Distributed key generation. Pedersen's protocol [17] lets a number of players generate an ElGamal public/private key pair in distributed fashion. At the end of the protocol, all players learn the public key. Each player learns a share x_i of the private key x such that $\sum_{i=1}^{k} x_i = x \mod q$.

Encryption/decryption. To encrypt $m \in G$, choose $r \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q^*$ and let $E(m) = (g^r, my^r)$. To decrypt a ciphertext (a, b), compute $b/a^x = m$. When the private key is shared among k players, the value a^x is computed jointly as $a^x = \prod_{i=1}^{k} a^{x_i}$.

Proof of correct encryption. This protocol allows a prover to convince a verifier that an ElGamal ciphertext E (created by the prover) is an encryption of a message m. The prover simply outputs m and the randomness r used to create the ciphertext E. The verifier checks that $E = (g^r, my^r)$.

Proof of Discrete Log equality. The Chaum-Pedersen protocol [7] lets a prover convince a verifier that a quadruplet (g, y, a, b) satisfies the equation $\log_g(y) = \log_a(b)$. Note that this protocol also proves correct decryption.

Multiplicative homomorphism. Let $m_1, m_2 \in G$, and let (a_1, b_1) and (a_2, b_2) be the ElGamal encryptions of m_1 and m_2. It is easy to verify that $(a_1 a_2, b_1 b_2)$ is an ElGamal encryption of $m_1 m_2$.

Oblivious test of plaintext equality. Let $E(m_1)$ and $E(m_2)$ be two ElGamal ciphertexts. A protocol of Jakobsson and Schnorr [13] lets the joint holders of the decryption key determine whether $m_1 = m_2$ without revealing any other information (hence the name oblivious). Using ElGamal's multiplicative homomorphism, the problem is equivalent to testing whether an ElGamal ciphertext (a, b) is an encryption of 1, i.e. whether $\log_g(y) = \log_a(b)$. Note that the Chaum-Pedersen protocol described above can not be used here because it leaks information when $\log_g(y) \neq \log_a(b)$.

Disjunctive Proof of equality of plaintext ($DISPEP$). A prover is given two ElGamal ciphertexts E_1 and E_2. The $DISPEP$ protocol [12] allows a prover to convince a verifier that an ElGamal ciphertext E is a re-encryption of either E_1 or E_2 without revealing which one. We refer the interested reader to [12] for a description of $DISPEP$.

Proving subset membership. Let E be an ElGamal ciphertext. The following protocol lets a prover (who knows the plaintext) convince a verifier that E is an encryption of a value in the set $\{g^0, \ldots, g^{51}\}$, without revealing which value.

- Let $T_1 = \{g^0, g^{26}\}$, $T_2 = \{g^0, g^{13}\}$, $T_3 = \{g^0, g^6\}$, $T_4 = \{g^0, g^3\}$, $T_5 = \{g^0, g^2\}$, $T_6 = \{g^0, g^1\}$ (We assume that the sets $T_1, \ldots, T_6$ have been precomputed). The prover outputs 6 ciphertexts $C_1, \ldots, C_6$ and proves with DISPEP that $C_i \in T_i$ for $i = 1, \ldots, 6$.
- The verifier decrypts $\prod C_i / E$ and verifies that the plaintext is 1.

Note that a very simple variant of this protocol allows a prover to convince a verifier that an ElGamal ciphertext E is an encryption of a value in the set $\{g^0, g^{52}, \ldots, g^{52(k-1)}\}$, without revealing which value.

Primitive	Cost
Encryption, re-encryption	2
Proof of DLog/decryption	
Prover	2
Verifier	4
Oblivious plaintext equality test	$8k-1$
$DISPEP$	
Prover	9
Verifier	4
Proving subset membership	
Prover	$64+4k$
Verifier	$22+4k$

In the table above and those that follow, recall that all costs are expressed as the number of modular exponentiations per player.

A. Group Establishment

While ElGamal's homomorphism would at first appear poorly suited to our application, we show how to make efficient use of it. Compared to dense probabilistic encryption, ElGamal offers the advantage of a simple distributed key generation algorithm.

Group establishment:

- The players use Pedersen's protocol [17] to generate a distributed ElGamal public/private key pair.
- Every pair or players (P_i, P_j) establishes a secret key $k_{i,j}$ for a symmetric cipher such as DES or AES. These keys allow any two players to communicate privately.
- The players precompute and store in memory the values $g^0, g^1, \ldots, g^{51} \in G$. These values encode the 52 cards.
- Recall that we let k denote the number of players. The players precompute and store in memory the following $k-1$ ElGamal ciphertexts: $D_i = E(g^{52i})$ for $i = 0, \ldots, k-1$. Let $S = \{D_0, \ldots, D_{k-1}\}$.

Group establishment	Cost
Pedersen protocol	$2k$
Generate secret keys	$2(k-1)$
Precompute cards	≈ 0
Precompute the set S	$2k$
Total	$\approx 6k$

B. Dealing the First Round

In our ElGamal implementation, cards in the first round are dealt differently from cards in subsequent rounds. We describe first how to deal the first round of cards. These cards are dealt either face up or face down, either to a player or in the middle of the table, as follows.

Generating a card:

1) Every player P_i chooses $r_i \overset{\mathrm{R}}{\leftarrow} \{0, \ldots, 51\}$, computes the ciphertext $C_i = E(g^{r_i})$ and outputs a non-malleable commitment to C_i. Note that a malicious player may choose $r_i \notin \{0, \ldots, 51\}$. It will be clear that such cheating is of no consequence.
2) Each player P_i reveals C_i, and all players verify that all commitments are correct. If one or more commitments are incorrect, the protocol aborts and honest players establish a new group that excludes the dishonest players.
3) Using ElGamal's homomorphism, the players compute $E(\prod_i g^{r_i}) = E(g^c)$, where $c = \sum_i r_i \mod q$. Note that if all players behave honestly, $c \in \{0, \ldots, 51k\}$.

Dealing a card face up (visible to everyone): Every player reveals r_i and the randomness used in generating $E(g^{r_i})$. The players verify that all $r_i \in \{0, \ldots, 51\}$ and that the encryptions are correct. If one or more players cheated, the protocol aborts and the honest players establish a new group that excludes the dishonest players. If all players were honest, the card dealt face up is $\sum_i r_i \mod 52$.

Dealing a card face down to player P_j: Every player reveals only to player P_j the value r_i and the randomness used in generating $E(g^{r_i})$ (the pairwise symmetric encryption keys set up during group establishment permit private communication between players). Player P_j verifies that all $r_i \in \{0, \ldots, 51\}$ and that the encryptions are correct. If one or more players cheated, the protocol aborts and the honest players establish a new group that excludes the dishonest players. If all players were honest, the card dealt face down to P_j is $\sum_i r_i \mod 52$.

Reducing cards dealt face down modulo 52. Assume that player P_j has received the ciphertexts $E(g^{r_i})$ for $i = 1, \ldots, k$. Let $c \in \{0, \ldots, 51\}$ such that $c = \sum_{i=1}^{k} r_i \mod 52$. Player P_j outputs the ciphertext $E(g^c)$. Player P_j must also output a proof that the value c has been correctly reduced modulo 52. The proof proceeds as follows:

- P_j proves that $E(g^c)$ is an ElGamal encryption of a value in the set $\{g^0, \ldots, g^{51}\}$. To do so, P_j uses the protocol to prove subset membership (described earlier).
- P_j proves that $E\big(g^{\sum_i r_i}\big)/E(g^c)$ is an encryption of a value in the set $\{g^0, g^{52}, \ldots, g^{52(k-1)}\}$. To do so, the player uses the variant of the protocol to prove subset membership.

Testing for "collisions" (cards dealt more than once). Collisions between cards dealt face up are trivial for anyone to detect. For collisions between one card dealt face up and one card dealt face down, or between two cards dealt face down, we use an oblivious test of plaintext equality (note that this works because all cards have been reduced modulo 52). If collisions are found, the colliding cards are discarded and new cards dealt in their place until there are no collisions.

A note on cheating. A malicious player (or several malicious players) may contribute a value $r_i \notin \{0, \ldots, 51\}$ for a card generated for an accomplice. If the accomplice fails to report that the value r_i is incorrect, this cheating goes undetected. However, this cheating is of no consequence: in particular, the card received by the accomplice remains exactly uniformly distributed.

Operation	Cost
Generating a card	2
Dealing a card face up/down	≈ 0
Receiving a card face up/down	$2(k-1)$
Reducing a card face down	
Prover	$< 2(64+4k)$
Verifier	$< 2(22+4k)$
Testing for one collision	$8k-1$

C. Dealing Following Rounds

Subsequent rounds differ from the first round in that collisions must be detected (at greater cost) before cards are ever dealt. Consider the following difference. When a collision between cards occurs in the first round, it is acceptable to discard both cards and start over. By contrast, it is no longer acceptable to eliminate both cards involved in a collision after the first round, since one of the cards involved in the collision may have been dealt to a player in a previous round and cannot be discarded without impacting that player's game.

Generating a card: Same as in the first round.

Mixing the set S: The players mix the set $S = \{D_0, \ldots, D_{k-1}\}$, using e.g. Millimix [12]. The cost of k players mixing t inputs with Millimix is $(3k+4)f(t)$ where $f(t)$, the number of comparitors, is given by $f(t) = t\lceil \log(t) \rceil - t + 1$ (see [12]). Note that here, Millimix is

used to mix a set of size $k \ll 52$ (in games of poker, typically $k \leq 6$) and thus this mixing operation is considerably faster than directly mixing the deck of 52 cards.

Testing for "collisions" (cards dealt more than once). If $L \neq \emptyset$, the players must test whether the newly generated card $E(g^c)$ already belongs to L. More precisely, for every ciphertext $E(g^{c'}) \in L$, the players must determine whether $c = c' \mod 52$. This is done as follows:

1) Using ElGamal's homomorphism, the players compute $E(g^c/g^{c'}) = E(g^{c-c'})$.
2) Using the oblivious test of plaintext equality, the players compare $E(g^{c-c'})$ with every ciphertext in S. If a match is found, the protocol aborts (the card c has already been dealt). In that case, the players start over and generate a new card as above (the set S must be mixed again).

If no equality is found, i.e. for all $E(g^{c'}) \in L$, we have $E(g^{c-c'}) \notin S$, then the card c has not yet been dealt in the current game.

Dealing a card: After ensuring there are no collisions, cards are dealt face up or down as in the first round (as in the first round, cards dealt face up need not be reduced).

V. Games

We consider the following games. Each game is played with a deck of 52 cards. Note that we are concerned only with the distribution of cards (and rounds of betting insofar as it may influence the number of rounds of communication).

- **Texas Hold'em:** Initially, two cards are dealt face down to each player, then there is a betting round. Three "community" cards are then dealt face up in the center of the table. Another betting round occurs. Another card is dealt face up in the center, followed by another betting round. Then a final card is dealt face up in the center, followed by a final betting round.
- **Seven Card Stud:** each player is dealt 7 cards, starting with two cards face down and one card face up. Three more cards are dealt to each player face up, with betting rounds in between. Then a final card is dealt face down, followed by a final betting round.

The following table shows the expected total real cost of a game with k players, measured in terms of the number of exponentiations that each player must perform. The first column shows the cost of a mixnet solution with Millimix (described in related work in Section I-B). The second column shows the cost of our protocol implemented with Dense Probabilistic Encryption or DPE (see Section III). Finally, the third column shows the cost of our protocol implemented with ElGamal (see Section IV). The costs are given for a game of Texas Hold'em with respectively 3 and 5 players. Very similar results are obtained for Seven Card Stud.

The DPE implementation of our protocol is by far the most efficient, but it relies on a trusted third party for initial key establishment (this may be acceptable since the trusted third party is used only once to set up a group). Our protocol implemented with ElGamal (for which no trusted third party is ever needed) offers smaller but still substantial savings compared to generic shuffling techniques based on mixnets. Furthermore, the computational cost is distributed more evenly across the game.

Scheme	Millimix	DPE	ElGamal
Texas Hold'em (3 players)			
Deck setup & 1st round	4233	201	≈ 800
Additional rounds (Max)	30	298	≈ 660
Texas Hold'em (5 players)			
Deck setup & 1st round	6239	1088	≈ 1700
Additional rounds (Max)	42	858	≈ 2900

VI. Conclusion

We have proposed a new protocol for shuffling and dealing cards, that is designed specifically for games of mental poker. Compared to generic protocols for shuffling cards, our approach offers a dramatic decrease in latency and overall computational cost. Our protocol is ideally suited for resource-constrained devices. We described our protocol for games of Poker but simple variations could be designed for Blackjack and other games.

References

[1] Rules of poker. `http://www.gambling-poker.com/`.

[2] B. Arkin, F. Hill, S. Marks, M. Schmid, T. J. Walls, and G. McGraw. How we learned to cheat at online poker: A study in software security. `http://www.developer.com/tech/article.php/10923_616221_1`.

[3] A. Barnett and N. Smart. Mental poker revisited. In *Proceedings of Cryptography and Coding*, pages 370–383, 2003.

[4] J. Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas in Cryptography 1994*, pages 120–128.

[5] D. Boneh and M. Franklin. Efficient key generation of shared RSA keys. In *Proceedings of Crypto '97*, pages 425–439.

[6] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, Feb 1981.

[7] D. Chaum and T. Pedersen. Zero-knowledge undeniable signatures. In *Proceedings of Eurocrypt '90*, volume 740 of *LNCS*, pages 89–105.

[8] C. Crépeau. A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face. In *Proceedings of Crypto '86*, volume 263 of *LNCS*, pages 239–247.

[9] J. Edwards. Implementing electronic poker: A practical exercise in zero-knowledge interactive proofs. *Master's thesis, Department of Computer Science, University of Kentucky*, 1994.

[10] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, Jul 1985.

[11] Y. Frankel, P. MacKenzie, and M. Yung. Robust efficient distributed RSA-key generation. In *Proceedings of the thirtieth annual ACM Symposium on Theory of Computing*, pages 663–672.

[12] M. Jakobsson and A. Juels. Millimix: Mixing in small batches, 1999. DIMACS Technical Report 99-33.

[13] M. Jakobsson and C. Schnorr. Efficient oblivious proofs of correct exponentiation. In *Proceedings of Communications and Multimedia Security*, pages 71–86, 1999.

[14] K. Kurosawa, Y. Katayama, W. Ogata, and S. Tsujii. General public key residue cryptosystems and mental poker protocols. In *Proceedings of Eurocrypt 1990*, volume 473 of *LNCS*, pages 374–388.

[15] D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Symposium on Computer and Communications Security*, pages 59–66. ACM Press, Nov 1998.

[16] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *Information and Communications Security '97*, volume 1334 of *LNCS*, pages 440–444.

[17] T. Pedersen. A threshold cryptosystem without a trusted third party. In *Proceedings of Eurocrypt '91*, volume 547 of *LNCS*, pages 129–140.

[18] C. P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.