Games for Extracting Randomness

Ran Halprin and Moni Naor

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
{ran.halprin, moni.naor}@weizmann.ac.il

ABSTRACT

Randomness is a necessary ingredient in various computational tasks and especially in Cryptography, yet many existing mechanisms for obtaining randomness suffer from numerous problems. We suggest utilizing the behavior of humans while playing competitive games as an entropy source. This idea has two motivations: (i) results in experimental psychology indicate that humans are able to behave quite randomly when engaged in competitive games in which a mixed strategy is optimal, and (ii) people have an affection for games, and this leads to longer play yielding more entropy overall. While the resulting strings are not perfectly random, we show how to integrate such a game into a *robust pseudo-random generator* that enjoys backward and forward security.

We construct a game suitable for randomness extraction, and test users playing patterns. The results show that in less than two minutes a human can generate 128 bits that are 2^{-64}-close to random, even on a limited computer such as a PDA that might have no other entropy source.

As proof of concept, we supply a complete working software for a robust PRG. It generates random sequences based solely on human game play, and thus does not dependent on the Operating System or any external factor.

1. INTRODUCTION

1.1 The Importance of Being Random

Randomness is essential for addressing the main problems of Cryptography (encryption, identification and authentication)[1], and it is also used in various other fields of Computer Science (see for example [1], [2], [3]) as well as other sciences (see [4]). When designing random algorithms and cryptographic systems, the availability of a source of pure randomness is assumed, but such a perfect source of randomness is not known to exist. To overcome this in applications the common approach is to find a source assumed to be of high min-entropy (see Definition 2.1), or at least unpredictable from the point of view of an adversary, and then extract a uniform (or close to uniform) sequence from it (see Section 2.1 for more about randomness extraction). This randomness source usually consists of a combination of information assumed to be unknown to an adversary (e.g. components manufacturer ID) combined with information assumed to be difficult to predict (e.g. exact system time or mouse position/movement).

While in principle this is a good approach, the implementation of such an extraction method is often problematic. Ferrenberg et al. demonstrated how certain choices of such an extraction procedure could lead to correlated sequences, eventually resulting in a random algorithm giving wrong results [5]. Early on in the history of the web Goldberg and Wagner attacked Netscape's implementation of the SSL protocol, basing their attack on its entropy sources and refresh rate [6]. Even more than a decade later, security vulnerabilities are still found in Pseudo-Random Generators: Gutterman, Pinkas and Reinman found weaknesses in the Linux Pseudo-Random Generator entropy collection stage in 2006 [7], and a bug in the Pseudo Random Generator of the Debian Linux Operating System (including the popular Ubuntu Linux distribution) caused generated cryptographic keys to be insecure. Shacham and Enright showed that a meaningful amount of keys generated during this period are highly predictable [8]. Alarmingly, this bug was only discovered in May 2008, two years after being introduced into the system [9].

These problems and pitfalls in Pseudo-Random Generation call for careful improvements in the design of such systems, and for better sources of entropy to be utilized.

1.2 Entropy Gathering

Systems that exist today utilize one or more out of three entropy collections methods:

1. **Direct Gathering** - the user is requested to hit the keyboard or wiggle the mouse randomly for a period of time. PGP is a well known example of direct entropy request [10]. The main problem with this method is that humans click the keyboard or move the mouse in relatively predictable patterns. Even worse, the casual user tends to perform very similar patterns every time they are prompted, leading to a very predictable result.
2. **Background Gathering** - the system constantly col-

[1]Under Kerckhoffs' principle (that states that the key is the only parameter of a system that can be considered secure), any deterministic computation by the parties to determine the key can be simulated by the adversary.

lects entropy from "least-significant data" - usage patterns of the mouse, keyboard, hard disk, etc. This method is very popular and is used in the Linux and Windows based systems [11] [12]. A problem with this solution is that it constantly spends resources on this collection. More disturbingly, the usage patterns might be predictable, for instance if the adversary convinces the user to perform certain tasks, or if the system is not directly used for a long period of time. Also, this method is not feasible when dealing with simple systems such as PDAs (Personal Digital Assistants, i.e. relatively weak handheld computers) that don't have a keyboard or hard disk, or in systems that are not frequently used.

3. **Hardware Gathering** - the system observes chaotic systems such as cloud patterns, electrical circuits, atmospheric noise and even Lava lamps [13] [14] [15]. Such systems have large overhead, are physically complex and therefore expensive to create and maintain. Also, it is relatively easy to interfere with the results, either by mistake of the user or by an adversary. The camera could be faced towards the wall, the lava lamp could be disconnected, and so on. In addition, the adversary may be able to observe the same system independently of the user and predict much of their entropy, and there is always the danger of temporary predictable behavior: Cloudless days, strong radio transmissions that dominate atmospheric noise, etc.

1.3 Humans and Randomness

The ability of humans to choose randomly has been extensively studied in Experimental Psychology (see [16] for a historical survey). It has been generally accepted that sequences and numbers generated by humans are far from being truly random[2]. Common biases of randomness **recognition** such as the "Hot Hand" (tendency to believe that a winning streak will usually continue), the inverse "Gambler's Fallacy" (tendency to believe that after a losing streak, the next attempt is more likely to be a success) and the related "Flip Bias" (tendency to believe 0 is likely to be followed by 1 and vice versa) have been thoroughly studied (and shown, statistically, to be fallacies) [18] [19] [20].

"Flip Bias" was shown to exist in randomness **generation** as well [21]. The result extends to digit generation as well. Figurska et al. showed, in a recent 2008 study, that humans tend to choose successive identical digits with probability 7.58% instead of 10% [22][3]. Therefore it was not surprising that humans assessed human-generated sequences (created by other humans asked to generate sequences that they would expect from multiple coin tosses) as more random than truly-random sequences, actually generated by coin tosses [23].

An interesting twist in the study of humans and randomness emerged when Budescu and Rapoport demonstrated that under some conditions, namely when participating in the zero sum competitive game "Matching Pennies" in which each player chooses a red or black card, the first player wins if both players choose a different card and the second player wins if the cards are identical. When playing this game, the players have an incentive to behave randomly, and human choices were shown much closer to random than when they attempt to simulate a random process [24].

It is Only Human to be Biased: While better than expected, human's gameplay patterns are still not perfectly random. Eliaz and Rubinstein showed that when playing the Matching Pennies game, the guesser (a.k.a. seeker) seems to have a slight empirical advantage over the misleader (a.k.a. hider) [25]. Specifically, guessers had a winning rate of up to 0.542 (rather than 0.5 expected.)

A more disturbing bias in the context of our work is the tendency of winning guessers to repeat their choice (60.7%) and the tendency of winning misleaders to change their choice after success (57.7%). We remind the reader that this bias was in a two-choice game, and its extension to an n choice game is not trivial.

1.4 Goals and Contributions

Our main goal is to present the idea of using human gameplay as a randomness source for cryptographic purposes. There are two lines of reasonings behind this idea: (i) The competitive nature of the game may make human act more randomly when playing games (than, say, when just asked to act randomly), as was demonstrated in an experiment by Rapoport and Budescu [24]. (ii) Playing games is more entertaining to users than simply "supplying entropy", meaning they will probably be willing to participate in the process and supply more data. A similar reasoning to (ii) was used by von Ahn who constructed games to motivate people to perform hard AI tasks and showed that (some) people are willing to invest much of their time in games [26].

Human game play is not perfectly random and Cryptography requires randomness that is very close to uniform[4]. Using randomness extractors (see Section 2) we can get very good random strings from longer strings that may be biased, provided they have sufficient entropy. This means that by motivating humans to supply longer sequences, the result of playing the game, we can improve the randomness of the result (the fact that they are playing a game means that they are willing to supply longer sequences than otherwise).

In other words, reason (i) above means that that people will supply a 'better' random string (more entropy per bit) and reason (ii) means that they will supply a longer string (overall more entropy).

Not all games are suitable for randomness extraction and we discuss the properties that games should have in order to be useful for our purposes. When choosing a game, it is important that we have an estimate on the quality of randomness humans supply when playing it. We suggested and implemented a specific game (Hide and Seek) and analyzed the biases humans show when playing it and checked the number of good random bits that can be extracted from it.

Once we have good random string of sufficient length, we can generate arbitrarily long pseudo-random sequences, which are sufficiently random for Cryptographic use. However, throughout the lifetime of a system the secret random-

[2]Humans asked to choose a number between 1 and 10 usually choose 7. For the range 1 to 20, they usually choose 17 [17]. Similar biases are known for other ranges.

[3]The study also shows that when humans generate a set of digits, they choose pairs of the form $(a, a+1)$ and $(a, a-1)$ with over 32% (expected 18%). This might be a basis for a generalization of flip bias.

[4]E.g. imagine a onetime pad where is each bit is slightly biased towards 1, and the same message is repeatedly encrypted; a cryptanalyst can easily figure out the message by taking the majority of each bit position

ness may leak or become corrupted. We show how to integrate our randomness extractor from human game play with a robust PRG system, as suggested by Barak and Halevi [27] for such purposes; the scheme enjoys 'backward' and 'forward' security. We implemented a complete system that is independent from the Operating System and does not require hardware access such as an exact clock, but rather utilizes only human playing patterns to generate randomness.

The organization of this paper is as follows:

In Section 2 we describe the combinatorial (randomness extractors) and cryptographic (pseudo-random generators) tools that we employ.

In Section 3 we discuss the desired properties of games used to extract randomness and the challenges in meeting them.

In Section 4 we design and implement a specific game and show the feasibility of extracting sufficient randomness from human playing patterns.

In Section 5 we describe the implementation the integration of the game with a complete robust PRG system that requires no randomness sources except for game play.

In Section 6 we discuss possible ways to improve the system, other games that can be use and suggest future work on this topic.

2. PRELIMINARIES

2.1 Randomness Extraction

The earliest discussion of "weak" random sources was probably made by von Neumann [28] who considered the case of a biased source that outputs a 0 with probability p and 1 with probability $1-p$. In recent time the theory of randomness extractors has been thoroughly investigated and methods have been found to extract from sources without a known structure and where the only requirement is *high min-entropy*. The discussion below is adapted from the survey by Shaltiel [29].

DEFINITION 2.1. *The **min-entropy** of a distribution X on $\{0,1\}^n$ is defined by:*

$$H_\infty(X) = \min_{x\in\{0,1\}^n} \log_2 \frac{1}{\Pr[X=x]}$$

In other words, if X has min-entropy k, then the probability of any single element to be drawn from X is at most $\frac{1}{2^k}$. Intuitively, although the distribution is over n bits, it "contains" enough randomness only for k uniformly distributed bits, meaning we cannot expect to extract more and the question is how close to k can we obtain.

DEFINITION 2.2. *Two distributions P and Q over the same domain T are **statistically ε-close** if:*

$$\frac{1}{2}\sum_{x\in T} |P(x) - Q(x)| \leq \varepsilon$$

Intuitively, if a distribution is ε-close to uniform we can use it instead of uniform with "damage" of ε.

An extractor is a function that takes a biased random stream and transforms it into a near-uniform random stream. Formally:

DEFINITION 2.3. *A (k,ε)-**extractor** is a function*

$$Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

such that for every distribution X on $\{0,1\}^n$ with $H_\infty(X) \geq k$, the distribution $Ext(X,s)$ (where $s \in_R \{0,1\}^d$, i.e. the seed s is d bits randomly chosen from the uniform distribution over $\{0,1\}$) is statistically ε-close to the uniform distribution on $\{0,1\}^m$.

While extractors can be used in cryptographic systems, the seed of extraction might become known to the adversary (such as when the system's randomness is temporarily compromised). If the extractor's output is still close to random even when given the seed, we call the extractor "strong".

DEFINITION 2.4. *A (k,ε)-**strong extractor** is a function Ext as above, where the distribution $Ext(X,s) \circ s$ (where $\circ$ represents concatenation) is statistically ε-close to the uniform distribution on $\{0,1\}^{m+d}$.*

Note that using a strong extractor implies that the seed is reusable.

An example of a strong extractor is based on families of pairwise independent hash functions. In general, ℓ-wise independent hash functions are functions such that $\forall x, h_s(x)$ is uniformly distributed over the random choice of s, and where $\forall j \in \{1,2,...,\ell\}$ the random variables $\{h_s(x_j)\}_{x_j\in\{0,1\}^n}$ are independent. A construction of ℓ-wise independent hash functions can be based, for example, on random polynomials of degree $\ell-1$. The extractor is defined by the following theorem:

THEOREM 2.1. *Let $H = \{h_s\}$ be a family of pairwise independent hash functions $h_s : \{0,1\}^n \to \{0,1\}^m$. For every $\varepsilon > 0$ and a randomly chosen seed s, the function $Ext(x,s) = h_s(x)$ is an $(m + 2\log_2 \frac{1}{\varepsilon}, \varepsilon)$-strong extractor.*

In other words, given a source of n bits with k bits of min-entropy, for any choice of $\varepsilon > 0$, applying a pairwise independent hash function $h_s : \{0,1\}^n \to \{0,1\}^m$ where $m = k - 2\log_2 \frac{1}{\varepsilon}$ results in m bits that are ε-close to random.

2.2 The Barak-Shaltiel-Tromer Model

The classic extraction model assumes that the source distribution from which extraction is performed is chosen independently from the extraction seed. However, in some settings an adversary might have some influence on the distribution. For example, in our setting the adversary might be able to choose which user to attack (different users have different biases), and perhaps they also have some limited influence on the user's playing patterns.

Such a setting was discussed by Barak, Shaltiel and Tromer in [30]. Their model allows the extraction procedure to be resilient against limited adversarial influence on the entropy source *after* the public randomness was chosen and as a function of it. Similarly to Definition 2.1, we have an extractor Ext, a seed s and an input x, but the way these elements are chosen is slightly different. The security of the model is defined by the following process:

1. An adversary chooses 2^t distributions $D_1, D_2, ..., D_{2^t}$ over $\{0,1\}^n$ under the constraint that $H_\infty(D_i) > k$ for $1 \leq i \leq 2^t$.
2. A seed $s \in \{0,1\}^d$ is chosen at random.

3. The adversary is given s and selects $i \in \{1, ..., 2^t\}$.

4. x is drawn from D_i and the user computes $Ext_s(x) = Ext(x, s)$.

In this setting, we can define t-resilient extraction:

DEFINITION 2.5. *An extractor is called t-resilient (with parameters ε, k, n, m) if for any adversarially chosen 2^t distributions as above, with probability $1 - \varepsilon$ over the choice of the seed s, if X is distributed according to one of the above D_i then $Ext_s(X)$ is statistically ε-close to the uniform distribution on $\{0,1\}^m$.*

Barak et al. show that pairwise independent hash functions are t-resilient extractors with certain parameters, and that ℓ-wise independent hash functions can achieve even better resilience, to some extent, as ℓ grows.
The extractor itself is as follows:

1. Select a random seed $s \in \{0,1\}^{\ell n}$ specifying h_s of a family of ℓ-wise independent hash functions $\{h_s\}_{s \in \{0,1\}^{\ell n}}$.

2. Given $x \in \{0,1\}^n$, return $E_s(x) = h_s(x)$.

Barak et al. show the following tradeoffs exist between t-resilience and the other extraction parameters:

THEOREM 2.2. ***(pairwise independent)*** *For every n, k, m and ε, a pairwise independent hash-function with a seed of length $d = 2n$ is a t-resilient extractor such that*

$$t = \frac{k-m}{2} - 2\log_2(\frac{1}{\varepsilon}) - 1$$

(ℓ-wise independent) *For every n, k, m, ε and $\ell \geq 2$, an ℓ-wise independent hash-function with a seed of length $d = \ell n$ is a t-resilient extractor such that*

$$t = \frac{\ell}{2}(k - m - 2\log_2(\frac{1}{\varepsilon}) - \log_2 \ell + 2) - m - 2 - \log_2(\frac{1}{\varepsilon})$$

We emphasize again that the only requirement for secure extraction in this model, similarly to the classic model, is high min-entropy. Higher min-entropy immediately implies higher resilience.

2.3 Pseudo-Random Generators

A **Cryptographic Pseudo-Random Generator** is a computable function that, given a random seed, creates a longer pseudo-random sequence. More formally, a PRG is a deterministic polynomial time function $G : \{0,1\}^m \rightarrow \{0,1\}^n$ where $n > m$ so that for all Probabilistic Polynomial-Time algorithm A, $x \in_R \{0,1\}^m$ and $y \in_R \{0,1\}^n$, the difference $|Prob(A(G(x)) = 1) - Prob(A(y) = 1)|$ is negligible[5].

PRGs are extremely important in Cryptography, as they imply that a relatively short random seed can supply all the randomness needed for a system without significant security loss relative to a one-time pad.

While a PRG can be used to expand a short random sequence into a longer sequence, there are still considerable security vulnerabilities for real PRG systems. If an adversary manages to retrieve the seed used to generate the sequence, the entire sequence becomes predictable. Even partial information about the seed could be used by an adversary to perform educated guesses, and even short pieces of the generated pseudo-randomness could be used to verify these guesses.

[5]A function $f(n)$ is said to be negligible if $\forall poly(\cdot)\ \exists n_0$ such that $\forall n > n_0, f(n) < \frac{1}{poly(n)}$. In other words, the inverse of the function grows faster than any polynomial.

The Barak-Halevi Robust PRG

In order to address this problem, Barak and Halevi introduce the concept of a **Robust Pseudo-Random Generator** [27]. A robust PRG, as they defined it, is a system that is similar to a PRG with an input function *refresh()* that refreshes its entropy, and an output function *next()* that returns a pseudo-random sequence. A robust PRG fulfills the following properties:

- Forward security - assume the adversary learns the internal state of the system at a certain point in time. The past outputs of the system (as well as past states), generated prior to the break-in, should still be indistinguishable from random to the adversary.

- Backward security (break-in recovery) - assume the adversary learns the current state S. Following the next refresh (after the break-in), all outputs should be indistinguishable from random to that adversary.

- Immunity to adversarial entropy - assume the adversary gains complete control over the refresh entropy (but has no knowledge of the internal state of the system which has been previously refreshed). The output of the system should still be indistinguishable from a random sequence to that adversary.

A big advantage of robust PRGs is the ability to combine independent sources of entropy for a proven increase in security. This option can also be seen as a motivation for this work and for finding other sources of entropy as different as possible from those in use today.

Barak and Halevi construct a provably robust PRG from any cryptographically secure PRG G.

3. EXTRACTION FROM A HUMAN SOURCE

In the previous section we established that given a good (high min-entropy) short sequence, we can generate very long Pseudo-Random sequences. We also saw that given several different entropy sources, we can combine them for an even safer system.

In this section we analyze the settings in which human game play could be used as an entropy source,

3.1 Games Used for Extraction: Desiderata

In Section 2.1 we defined Randomness Extractors. We saw that the amount of randomness that can be extracted from a source is bounded by its min-entropy. Therefore, if we want to extract randomness from a human's playing patterns in a game, we want two major criteria to hold:

1. The game encourages humans playing it to use high min-entropy.

2. There exists a method to bound from below the min-entropy used by the player in an observed interaction, i.e. to recognize very low-entropy play when it is observed.

Regarding the first requirement, it is obvious that in order for the player's actions in the game to have large min-entropy, the player must have many options for each move (the number of possible moves and the number of "rounds" together bound the min-entropy from above). Also, we would want the players to have no reason to prefer specific moves over others, nor to prefer specific sequences of moves (i.e. have dependencies between the moves). We saw in Section 1.3 that human biases are increased when the mixed strategy is not uniform. Non-uniform mixed strategies are unnatural, and the higher bound for the min-entropy used by the players is lower. Therefore, we would prefer games with perfect (uniform) mixed strategy.

Regarding the second requirement, although we can deal with human biases to some extent (as discussed in Section 1.4), the guarantees on the quality of the results requires a bound on the min-entropy used by the player. An attempt to estimate the min-entropy of humans empirically is done in Section 4 and a suggestion for future work on the subject is given in Section 6.3.

In addition to the desired properties regarding min-entropy, there are also desired technical properties of such a game. The game should be reasonably short, so that players would not become annoyed and bored with playing it periodically. It also should be natural to play and not require extensive skills from the players, so that it could be used by people who do not usually play video games. In addition, the game should work on the most minimal of computing systems, and require no expensive or large hardware (e.g. high resolution screen or a fast processor). Not less important, the game should be at least somewhat interesting and entertain players long enough so that they will willingly play enough to produce many sequences. If the game is not entertaining, the player might become bored and resort to a "lets just get it over with" approach that might increase repetition and lower the entropy of the provided sequences.

3.2 Fitting a Game to the Desiderata

Consider the mixed strategy zero-sum game studied by Rapoport and Budescu in [24], Matching Pennies. The major problem with this game is that in each round the player faces only two options. This is very bad for our purpose, as a player produces at most one independent bit per round[6]. Modern Cryptographic keys, such as the ones used by AES are at least 128 bits long. Therefore, at least 128 rounds of this game need to be played in order to create a single key, making this game *inefficient* for our purposes.

A natural extension of Matching Pennies that enlarges the number of choices per move is the game **Hide and Seek** [31] [32]. In the classic version of Hide and Seek, one player (the hider) chooses a spot on a vector or grid (usually two-dimensional, but not necessarily) with n cells. The second player (the seeker) also chooses a spot on the same grid. The hider wins a point if the seeker chose a different cell, and the seeker wins a point if they both chose the same cell. Similar to Matching Pennies, the game is also a two player zero-sum perfect mixed strategy game. Assuming (very optimistically) that a player is perfectly random, for each choice in this grid they "Generate" $\log_2 n$ bits (obviously, in order to behave randomly, their min-entropy must be $\geq \log_2 n$ per click). Unless the grid is very large, one such game does not supply enough entropy for cryptographic uses. Even if the player is perfectly random, n would need to be at least 2^{128}, which is not a reasonable number of outcomes for a single human decision. In order to solve this, the game can be played over several rounds, until enough entropy was gathered. The methods to determine this amount is analyzed further in the next section.

An important observation is that similarly to Matching Pennies, if the hider's min-entropy is low they will lose more rounds than expected. This is because the seeker will be able to learn the patterns in their play and gain an advantage, so in the long run they will be punished for their low min-entropy.

The main challenge when attempting to construct a game to fit the desired properties is that while it should encourage people to play randomly, it should also be fun enough to play such that the players retain their interest and play long enough to produce enough entropy. These two properties are somewhat contradictory, as playing randomly does not normally appeal to humans. Popular recreational games usually challenge players to play accurately, which is in some sense the exact opposite of random play. In Section 4 we try to address this challenge as we design and implement a variant of "Hide and Seek".

3.3 Simulating One Player by the Computer

Rapoport and Budescu conducted their experiment in [24] on sets of two players, showing that game playing induces relatively random behavior. However, many systems for which we would like to generate randomness are private (e.g. a PDA), meaning two players are not always available. This naturally raises the scenario in which we simulate one of the players in the game by the computer. The human player should be able to play as if their opponent is human, as humans often do when they play single-player games against computer opponents.

in such a setting, the computer player should not be too predictable. If the human player is able to predict the computer player's behavior in advance, this would make the human playing patterns somewhat predictable and the human bored. We find ourselves forced to make its moves at least somewhat random, and this means that we need to "waste" some of our randomness.

An important point is that this randomness used by the computer player doesn't need to be so good. Rather, it should only be unpredictable enough to fool the *human* player who is naturally **limited** both computationally and in the information they have about the system. Any move pattern that is not trivial to detect is sufficient (but, of course, not to a malicious adversary who might be more sophisticated).

In the case of weak computers that lack entropy sources all together (no internal clock and no hard disk), some factory defined "randomness" could be pre-programmed into the machine, and later supplemented by small amounts of the human-extracted entropy when it is gathered.

Since the single-player setting is a very useful scenario (and apparently also the most challenging), we will concentrate on it throughout this work.

3.4 Games with Deterministic Strategies

While game theory often discusses games with mixed strategy, most recreational games actually played by people have

[6]Since humans are still not *perfectly* random, the real amount of entropy that can be extracted from this game is even smaller.

a deterministic optimal strategy, i.e. a finite sequence of moves that prevents losing in the game, or in the case of randomness based games (such as Backgammon or Monopoly), minimize the risk of losing[7]. This includes two-player games such as Checkers[8], as well as single-player video games such as Pacman[9] or Super Mario. These games rely on the player's skill in finding and performing winning sequences.

Games with an optimal strategy are obviously less useful for entropy extraction. Players often learn the basics of a game quickly and devise some basic templates that they use throughout the game. Nonetheless, it is still possible to extract little entropy from such game play by concentrating on least-significant behavior (i.e. low order bits). The exact position of a mouse cursor or the exact millisecond timing of clicks are not crucial to the success of the player in the game. Mixed Strategy is therefore not mandatory for extracting randomness.

A benefit of using such games is that we can choose an already popular game to collect entropy from, a game that is proven to attract players to play more. Analyzing the min-entropy for popular games, such as Tetris (which is specifically interesting because it is not deterministic), could prove interesting and useful. While embedding randomness extraction into existing popular games is practical and worth mentioning, it is beyond the scope of this work to analyze this further.

4. OF MICE AND ELEPHANTS

In this section we suggest a game suitable for randomness extraction and test the ability of humans to play it with high min-entropy. We then show the feasibility of extracting from the playing patterns sufficient randomness for common cryptographic applications.

Note that this is not a formal scientific experiment in the rigorous sense.

4.1 Constructing a High Entropy Game

Following the discussion in Section 3.2, we chose the game "Hide and Seek" as the basis for our experiment. In the design of our variant, we attempted to satisfy the desiderata (Section 3.1), i.e. we try to implement an entertaining game that encourages players to play randomly.

A problematic aspect of the Hide and Seek game is that similarly to Matching Pennies, it does not seem to be very entertaining[10]. In order to make this game more interesting and enjoyable so that humans will want to play it for the required length of time, we used lively mouse and elephant characters, we renamed it "Mice and Elephant" and incorporated animations and sound effects.

Instead of having discrete choices over a set of cells, the players choose a position in a seemingly continuous board. In the implementation the board is of course discrete, slightly larger than 512×256 pixels. This fits the screen nicely in any commonly used screen resolution, as well as produces approximately $9 + 8 = 17$ bits of raw data per click (note that human behavior in the game is not uniform, so the real randomness extracted from it will be smaller). Humans are not very accurate in their click positions, so the high-level entropy from the human choices is mixed with the low-level entropy from their inaccuracy.

In each round, the hider positions the mice on the grass and the seeker positions the elephant. After both players are done, a short animation sequence shows the elephant falling on the spot where it was placed. The game ends when the seeker's elephant "crashes" one of the hider's mice (i.e. it is positioned close enough to it). This brutal immediate end of the game encourages the player to play carefully, as well as adds an element of suspense. The animation makes the purpose of the game quite clear, even if the user did not read the instructions.

In our implementation, the mice are controlled by the human player (hider). The elephant is controlled by the computer player (seeker). We let the hider control r mice (where r is the round number), while the seeker controls a single elephant[11]. The objective of the hider is to survive as many rounds as possible, while the objective of the seeker is to catch a mouse as early as possible, ending the game.

The progression in the number of mice was chosen mainly in order to make the game more interesting. Many rounds of one mouse vs. one elephant would quickly bore the players - when all rounds feel the same, there is no feeling of advancing. The increasing probability for losing in each round builds up even more tension in the game, especially when the player approaches his previous highest round (For a discussion of this "Narrative Tension" and other considerations in game design, see [35]). An important effect of this choice is that it allows good players to introduce more entropy into the system - a more random player will generally reach higher rounds (as the opponent will find it harder to predict their behavior).

A problem immediately evident in the game as defined is that the best strategy is placing all mice in the same spot. Since the game ends when any mouse is hit and this strategy minimizes the chance of a mouse being caught. In order to prevent players from using this strategy, we prevent mice from being positioned too close to each other. While this slightly lowers our entropy gathering, it prevents the lazy strategy which might have very bad impact.

Another problem is players who use the natural tactic of repeating the same moves over and over from round to round. A simple solution was having the elephant, with high probability, choose its moves uniformly from the player's recent history of moves (once the player builds such a history, before that the moves are always random). This ensures that playing a certain region too often would cause the player to be caught faster. Another deterrent to any regional bias a player might have was added in the form of **obstacles**, marked areas where mice cannot be positioned. The system introduces several obstacles to the playing field that graphically convey to the player that a mouse cannot be positioned there: A big rock, a lava pit, a shark pool or a mousetrap. Similarly to the elephant's moves, the positions

[7] A near-optimal almost deterministic strategy for a certain Poker variant can be found in [33]

[8] In Checkers, perfect play by both players leads to a draw [34]. Thus, an optimal strategy for **winning** the game does not exist.

[9] An optimal strategy for the original Pacman game can be found at http://www.geocities.com/mamehelp/pacman.html

[10] When conducting their experiment Rapoport and Budescu paid 91 test subjects to play Matching Pennies, as customary in Social Sciences.

[11] Note that using more than one mouse per round does not promise an increase in the min-entropy needed to avoid the elephant.

of the obstacles are chosen uniformly from the player's previous moves. If players choose a certain region more often, this region would be more likely to be blocked by an obstacle. In the long run, this technique pushes the distribution towards uniform. Figure 1 shows the game as given in the final version.

Figure 1: Mice and Elephant with Obstacles

In order to allow the players to follow their advancement in entropy gathering, the players are given a score for each mouse, the number of obstacles (to compensate for the lowered number of options), the distance from the elephant and the time. The time bonus was mainly introduced to prevent the players from carefully planning patterns while positioning mice. In addition, Fitts' law (see [36]) predicts that the accuracy of clicks lowers when users have less time, which is to our advantage in this setting (accuracy is, in a sense, the opposite of randomness).

In the context of randomness extraction, it appears natural to demand that the player reaches a certain score before considering the entropy sufficient (see Section 5 for an analysis). Apart from ensuring the minimal amount of entropy sufficient for completely refreshing the system, having such a goal in mind should also make playing the game more interesting (A goal is perhaps the most important principle in a game, see [35]).

4.2 Experimental Setting

In order to test the typical human behavior in the game, we developed a web based game using Java 1.5, and publicized it at first to students in our department and later to the internet. The game is available online at the address **http://math166-pc.weizmann.ac.il/game**.

The experiment ran online for several weeks during which it was advertised on various internet forums, to friends and family, and in several academic gatherings.

It is important to note that most subjects were not aware of our objective, and were only instructed to attempt to mislead their opponent and survive as many rounds as possible.

4.3 Results Analysis

The map of all clicks played in the game by all 482 players are shown in Figures 2 and 3. The fact that these results are from many users (each contributing a different amount of data) makes it difficult to claim that this is the actually representation of a person's playing pattern. While there is too little data for analyzing individual players' patterns, we can still gain some insights from these results in regards to how humans play this game. In a sense, the Barak-Shaltiel-Tromer model justifies this approach, as it incorporates resilience to adversary influence (discussed in Section 2.2). Except perhaps for some extreme cases of incompetent users with very low min-entropy in their playing patterns, the result of extraction remains almost-uniform. This is true even when the adversary knows or affects some of the biases of the chosen user.

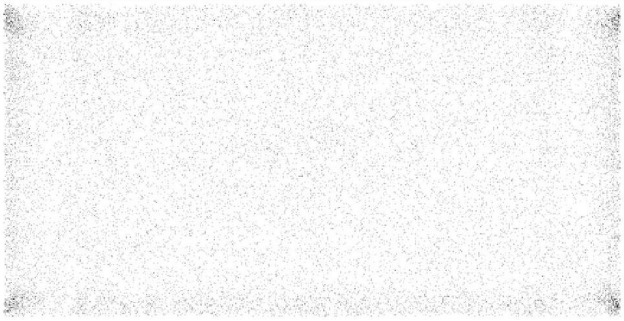

Figure 2: Click map over $24,008$ clicks by 482 users in the final game

Figure 3: Estimated distribution over $24,008$ clicks by 482 users in the final game

The results depict a high inclination for the corners and banks of the field[12], but in general the distribution is not far from uniform. Perhaps the most important result is the fact that the most highly represented single point in the playing field has only 7 clicks out of $24,008$ total clicks. This gives us an upper bound on the min-entropy of 11.74 per click. If all clicks were completely independent, this would have been a good estimate of the min-entropy.

Nonetheless, there is good reason to believe clicks are very much not independent. It is reasonable to assume that humans employ some strategy between clicks, especially within

[12]This is interestingly opposite to Rubinstein, Tversky and Heller's result for a four-cell Hide and Seek game, where players tended to avoid the endpoints [32]. A possible explanation that resolves this conflict in results would be that players perform a binary mental choice between "edge" or "not edge", and in our setting the amount of space that is perceived as an edge or corner is much smaller than in their experiment (where it is actually 50% of given options). The size of this mental "edge" is an interesting topic by itself.

a single round, where the mice are all visible on the screen as the same time. In order to test this, we visualize the differences between each click and the next click in Figure 4.

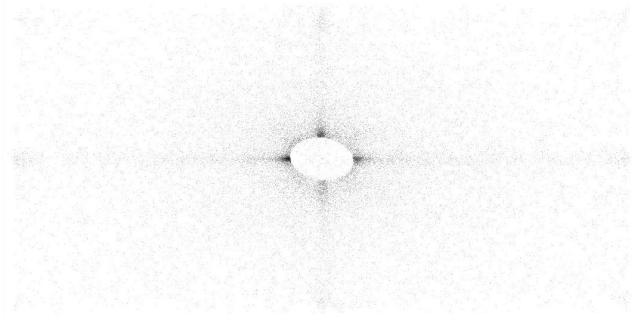

Figure 4: Click difference map over $24,008$ clicks (logarithmic scale)

The under-represented ellipse in the center of the map is the region where a click cannot be played (in the same round) nearby a previous click. Some clicks still exist in the range because the difference was measured also between rounds.

We notice that players have a tendency to click very close to the previous click. In the central rectangle of size 109×217 (which accounts for less than 4.1% of the complete map, and includes the central ellipse) there are 9145 clicks, accounting for over 40% of the clicks in the map. We also notice that players prefer playing on the same axis as the previous click, either left, right, up or down from a previous click. In the highly represented part on the left of the ellipse there is a rectangle of size 31×36 pixels (0.19% of the map) in which 1928 clicks have been introduced (more than 8% of clicks).

An important observation is that the maximal point represented in the map is 24 (out of $24,007$). This gives us an upper bound on the min-entropy of 9.96 per click, lower that the bound found on the click map. If we assume each click made in the game is dependent only on the previous click (i.e. that the process is Markovian), this could be a good estimate for the min-entropy per click, as the difference bias is much more dominant.

The second and third degree difference were also tested (given a click, the second order difference is the difference between a click and a click performed two clicks ago). The results were similar, but significantly less biased than the first order difference.

4.4 Statistical Result Analysis

The raw results as taken from the human play were concatenated to a single bit-string, ordered by users first and time later. On these results we ran a pattern distribution test We chose this randomness test mostly because it was used by Rapoport and Budescu to compare the statistical randomness in humans' playing pattern versus humans' attempts to generate random sequences in [24], and this analysis gives contrast to their results.

After taking the binary representation of the click locations and applying a 2-wise independent hash function (Multiplication in $GF(2^n)$) as an extractor. The extractor was applied on sequential tuples of 39 clicks ($39 \cdot 17 = 663$ bits) and a target size of $m = 128$ bits. According to Theorem 2.1, the results should be about 2^{-128}-close to uniform (without resilience), of course based on the assumption that the min-entropy of the data is 9.9 per click. The result was consistently (throughout five different random seed choices) good p-values (all under 0.95).

We should note that the subjects of this study were not instructed to play randomly, and some instruction on the matter could improve results even further, as discussed in Section 1.3.

5. A COMPLETE SYSTEM

5.1 Overview of the Proposed System

In this section we suggest a construction of a robust Pseudo-Random Generator based on human game play, that requires no other entropy source. The system is comprised of three major parts:

1. The game constructed in Section 4.
2. The extractor discussed in Section 2.2.
3. The Barak-Halevi Robust PRG discussed in Section 2.3.

The Barak-Halevi architecture is the skeleton of this system and the entropy source for it is the moves played in the game. The extractor used on this entropy is an ℓ-wise independent hash function with at factory set random seed. The user periodically refreshes the system with entropy extracted from the user's gameplay, and when requested, the system generates pseudo-random sequences for the user.

The system is started with some initial value as the internal state. The *refresh()* and *next()* operations utilize a cryptographic PRG implemented by using AES with the current state as the key.

5.2 The Detailed Construction

In this section we suggest parameter values for this system that fit current security practices and technologies. These values are, of course, not the only ones possible.

The first major choice is the block and state size (which determines the output size). For this, we chose 128 bits, which is sufficient for modern security requirements and fits modern cryptographic needs.

The system is imprinted with factory generated random sequences:

- $s \in \{0,1\}^{15,232}$ - The seed used for randomness extraction, must be chosen randomly. The value was chosen to suffice to extract using an $\ell = 32$-wise independent hash function, from blocks of 28 clicks or 476 bits.
- $\alpha_s, \alpha_n \in \{0,1\}^{128}$ - The constants used together as input to the PRG where the seed is the secret state. These values can theoretically be set to any arbitrary values such as 0 and 1, but random values add a layer of obscurity.
- $S_0 \in \{0,1\}^{128}$ - The initial state of the system. Can theoretically be set to any arbitrary value such as 0, but a random value adds a layer of obscurity.

These sequences can all be public without hurting the system's security. Their sizes were chosen as they require less

than 2 kilobytes of memory, which is a reasonable amount even for weak machines (one page of text would require more than 2kb). This settings allows generation of independent outputs of 128 bits, long enough for most cryptographic uses. Of course, repeated uses of *next()* allows users to get strings of any length, should the need arise.

The internal state is initially set to S_0. In order to start using the system, the user needs to refresh it at least once by playing enough rounds of the game. When at least 28 clicks have been played (80,000 points are enough to guarantee this with high probability), under the assumption that the min-entropy of the user is indeed 9.9 per click, this would ensure that a sequence with 277 bits of min-entropy are ready for extraction.

All data generated by the user is passed via an 32-wise independent hash function (as discussed in Section 2.2) with the seed s. The result is an $m = 128$ bit sequence. Under the assumption that the min-entropy is 9.9 and using Theorem 2.2, this sequence result is 2^{-64}-close to uniform with $t = 46$ adversary resilience.

At this point, the sequence is XORed with the current state S, and the result is given as the key to a block cipher implementation (AES is probably the best choice at the time of writing this) that encrypts α_s to get the new state.

After the system is set, it may be used to generate outputs. This is done by sending the current state as the key of a block cipher that encrypts α_n and returns the result as the output. The state is then immediately refreshed by sending it as a key to a block cipher that encrypts α_s.

5.3 Adapting to Incompetent Players

Under the assumption that the only significant biases a human user has are the click locations and differences, the min-entropy can be estimated for the specific user by analyzing their clicks and differences patterns. In order to assess the min-entropy, we can use a low-level filter on the click map and difference map, and check the maximal values in the maps. This allows to easily identify highly expressed regions or differences, in which case the estimated min-entropy would be lowered. If the min-entropy is estimated to be too low, the program can increase the amount of moves needed before generating the output, directly dependent upon Theorem 2.2. As the required entropy for refresh grows, the extraction rate is lowered, but security is never compromised.

5.4 Performance Analysis

In order to test the performance of the complete Human-Random Generator, we implemented a slightly simplified system using Java 1.5. S_0 was set to zero, z was not used, and extraction was done with a pairwise independent hash function, with a seed being a sequence arbitrarily chosen from [37]. Also, the number of mice per round was decreased to $\lfloor 1 + \frac{r}{2} \rfloor$ to prevent clutter.

Since the system uses only a pairwise independent hash function as an extractor, we require that the players reach 80,000 points which ensure that at least 28 clicks have been played. In this extreme case where $k = 28 \cdot 9.9 = 277$, the results are approximately 2^{-32}-close to random with $t = 9$ resilience. In practice, between 40 and 50 clicks have usually been played to achieve this score. With an average $k = 45 \cdot 9.9 = 445$, the results should be about 2^{-64} close to random with $t = 29$ resilience.

The implemented system is available online at **http://math166-pc.weizmann.ac.il/**. The system GUI is shown in Figure 5.

Figure 5: PRG System GUI - The user presses next as many times as desired to generate output strings.

As a second phase, we proceeded to test the randomness of this system by simulating it over long-term activation. We begin by initiating the internal state with a *refresh()* on a block of 128 bits extracted from the experiment results from Section 4.3. Similarly to the analysis in Section 4.4, extraction was made from blocks of 39 clicks from the experiment. According to Theorem 2.1, the results should be about 2^{-128}-close to random (without resilience).

The system then generated 10,240 sequential calls to *next()*, saving the resulting outputs sequentially into a file. The system was then refreshed with another block of human-extracted entropy. This pattern is repeated until there is no more source entropy. The extraction procedure resulted in slightly more than 64,000 bits, so we refreshed 500 times, and ended up with a result file of 614,400,000 bits. We tested this file using the DIEHARD statistical suite of tests [38] and found no significant statistical biases.

5.5 Combining Different Entropy Sources

A main advantage of the system constructed in this section, an inherent advantage of the Barak-Halevi architecture used, is that it can utilize several different entropy sources. The system is promised to be secure if at least one entropy source used is secure, and therefore using as many different entropy sources as possible can never harm security, but only increase it.

For example, one can consider initiating the system with an internal state derived from the system's constants (or in a PDA setting, perhaps a single long random code generated in the factory), and then refresh the system with human-extracted entropy periodically. A more complex combination could add an entropy gathering system in the background (similar to the systems mentioned in Section 1.2). The ongoing entropy gathering would be used to refresh the system in short intervals (e.g. minutes) while explicit extraction from the user via the game could be used in longer intervals (e.g. days).

6. CONCLUSIONS AND FURTHER WORK

In this work, we designed and implemented a game and analyzed the playing patterns of people who played it. Many people participated, implying that the game was reasonable

to play, e.g. not very difficult to understand. We saw that randomness extraction from this game is feasible, and that the randomness is enough to be used by a Pseudo-Random Generator with good empirical results. The option to combine this randomness with other entropy sources strengthens this result even more.

The main conclusion is that using human game play as a randomness source is a valid option, whether by itself when other forms of randomness are not possible. The possibility of combining this randomness with other entropy sources (see Section 5.5) makes this approach even more attractive.

6.1 Comparison With Direct Gathering

As mentioned in Section 1.4, one motivation of this work was that the competitive nature of games provokes humans to be more random. A natural question is then to see how significant is the game setting as opposed to, say, just asking users directly to click at random locations. Ideally, to establish the benefits of entropy gathering in a competitive setting, an experiment should be done to compare the results when playing the game and when requesting clicks directly from the user.

We actually attempted to conduct a similar experiment to the one in Section 4 and compare the biases in gameplay to the biases when directly asking for clicks (on an otherwise similar setting to the game and with the same restrictions). However, we had too few participants (around 50) who all played too little to provide us with meaningful results (they did not generate enough 'clicks' to allow extraction). This lack of motivation to participate in itself says something about the problems of direct gathering. Participants who were interviewed about their experience all said they resorted to some pattern, usually circles around the center, and attempted to "get it over with" as fast as possible.

This experience implies that it is worth studying the behavior of people in a more controlled environment as well (as opposed to just having the game on the web).

6.2 Dynamic Score Model and Obstacles

After observing the behavior of the players in Section 4.3 we reached the conclusion that the game can be improved to encourage more random game play. Since the first order difference contained a dominant bias, it is natural that we would want to counter it. One possible measure is to tweak the score model such that players who position mice in the highly represented areas, i.e. very close to a previous mouse or on the same X-axis or Y-axis, are punished by being awarded significantly lower scores, or even lose points. The reward for reaction speed might have encouraged players to play quickly and thus not move far enough, leading to the bias observed in the first order difference maps, so it remains to be tested how this affects the quality of the results (see Section 6.3).

A dynamic scoring process could be automated by superimposing the inverse of the cumulative click map and the difference maps as the score model. In effect, clicking a less common region or following a less common pattern would result in a higher score reversely proportional to the popularity of the point clicked. If the system is used by a single user, this model would encourage the user to be creative. Also, if the user is too repetitive and gains low score, the system would require more game play before refreshing. Considering the players' inability to be accurate, a low-pass filter (blur) of the cumulative click map could be used instead of the map itself, to promote playing less played areas. Leaving the users to learn the score model on their own is an additional challenge: The player might experiment to find the sort of behaviors that will grant them a higher score.

Similarly to the dynamic score mode, some form of dynamic obstacles can also be utilized. For example, when the player positions a mouse, the first order difference map for the user can be used to randomly position an obstacle relative to the user's last move. This could further assist in 'flattening' the playing patterns to being more uniform.

6.3 Additional Work

We suggest several directions for further research on this topic and related topics:

Different Games: In this work we studied a single game. Obviously, this is not the only possible game for our purposes and there are many other games that can be used. An interesting issue is trying to characterize the design rules that make people behave randomly.

Note that a complete system might include a set of different games that the user may choose to play. The option of different games would make the user more entertained, and adds another layer of obscurity for the adversary who does not know which game was played. One possible direction for more games is extensions of hide and seek such as Battleship, but it also possible to use very different games, e.g. Tetris.

A different direction to study is cooperative games. While in this work we only discussed competitive games, it is possible to create cooperative version of Hide and Seek, where choosing a different spot than the second player gives both players a higher score[13]. We also consider a cooperative game called the anti-ESP game[14], in which the two players choose an image or subset of images from a large set, and are rewarded for the difference in their choices. The inherent problem with cooperative games is that players wish to "communicate" a strategy to the other player, and therefore have an incentive to be predictable.

Different Populations: In the study performed in Section 4 we had no knowledge or control of the subjects involved, as it was completely anonymous. The difference of behaviors between different types of populations might change our key observations regarding the min-entropy of play. Some possible populations to compare could be older people vs. younger people, casual gamers vs. "heavy" gamers vs. non-gamers, males vs. females, and different cultures or countries.

Complete System Test: Our study of human playing patterns was done without a score threshold, but in our implementation the user knows when they are approaching the threshold of points. This knowledge and other factors could skew user behavior. Therefore, the system should be tested in its final version, over long term usage.

Human Accuracy and Fitts' law: Fitts' law predicts the time it will take a human to move a pointer to a tar-

[13]To a large extent, whether a game is cooperative of competitive is in the eye of the beholder and the way the game is framed.

[14]In the ESP game suggested by von Ahn and Dabbish in [39] two players attempt to agree on words that represent an image (without knowing each other's guesses until a match is found).

get area as a function of the distance to the target and the size of the target area (see [36]). It is often considered a good measure of the human speed-accuracy tradeoff (see, for example, [40]). By bounding the time we give a user to perform a task we therefore may predict that the accuracy of the click position would be damaged. If we consider randomness as the opposite of accuracy, Fitts' law gives us an estimate on the randomness generated by a human in a click task. We can use it as a basis to study the entropy of clicks against different time constraints and distances and derive estimated lower bounds on the entropy of human actions when performing certain tasks.

Acknowledgements

The authors would like to thank Anat Iloni for the graphic design of the implemented game and Tal Moran for helpful input and discussion.

7. REFERENCES

[1] R. Motwani and P. Raghavan, "Randomized algorithms," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 33–37, 1996.

[2] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer, 1999.

[3] S. Rajasekaran, *Handbook of Randomized Computing*. Kluwer Academic Publishers, 2001.

[4] J. Gentle, *Random Number Generation and Monte Carlo Methods*. Springer, 2003.

[5] A. Ferrenberg, D. Landau, and Y. Wong, "Monte carlo simulations: Hidden errors from "good" random number generators," *Phys. Rev. Lett.*, vol. 69, pp. 3382–3384, Dec 1992.

[6] I. Goldberg and D. Wagner, "Randomness and the netscape browser," *Dr. Dobb's*, pp. 66–70, January 1996.

[7] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the linux random number generator," in *IEEE Symposium on Security and Privacy*, pp. 371–385, IEEE Computer Society, 2006.

[8] H. Shacham and B. Enright, "The debian openssl bug and its effect on ssl." http://www.usenix.org/events/sec08/wips.html, 2008.

[9] F. Weimer, "New openssl packages fix predictable random number generator." http://lists.debian.org/debian-security-announce/2008/msg00152.html.

[10] P. Zimmermann, "Pgp(tm) user's guide." ftp://ftp.pgpi.org/pub/pgp/2.x/doc/pgpdoc1.txt.

[11] T. de Raadt, N. Hallqvist, A. Grabowski, A. Keromytis, and N. Provos, "Cryptography in OpenBSD: An Overview," *Proc. of the 1999 USENIX Annual Technical Conference, Freenix Track*, pp. 93–101, 1999.

[12] L. Dorrendorf, Z. Gutterman, and B. Pinkas, "Cryptanalysis of the windows random number generator," in *ACM Conference on Computer and Communications Security* (P. Ning, S. D. C. di Vimercati, and P. F. Syverson, eds.), pp. 476–485, ACM, 2007.

[13] M. Haahr, "Random.org - true random number service." http://www.random.org.

[14] G. Bernstein and M. Lieberman, "Secure random number generation using chaotic circuits," *Circuits and Systems, IEEE Transactions on*, vol. 37, no. 9, pp. 1157–1164, 1990.

[15] L. Noll, R. Mende, S. Sisodiya, *et al.*, "Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system." US Patent 5,732,138, 1998.

[16] W. Wagenaar, "Generation of random sequences by human subjects: A critical survey of literature," *Psychological Bulletin*, vol. 77, pp. 65–72, 1972.

[17] D. Munger, "Is 17 the "most random" number?." scienceblogs.com, specifically http://scienceblogs.com/cognitivedaily/2007/02/index.php?page=3, Feburary 2007.

[18] R. Falk and C. Konold, "Making sense of randomness: Implicit encoding as a basis for judgment," *Psychological Review*, vol. 104, pp. 301–318, April 1997.

[19] B. Burns and B. Corpus, "Randomness and induction from streaks: "gambler's fallacy" versus "hot hand"," *Psychonomic Bulletin and Review*, vol. 11, pp. 179–184, 2004.

[20] A. Tversky and T. Gilovich, "The "Hot Hand": Statistical Reality or Cognitive Illusion?," *Preference, Belief, and Similarity: Selected Writings*, 2004.

[21] A. Rapoport and D. Budescu, "Randomization in individual choice behavior," *Psychological Review*, vol. 104, no. 1, pp. 603–617, 1997.

[22] M. Figurska, M. Stańczyk, and K. Kulesza, "Humans cannot consciously generate random numbers sequences: Polemic study," *Medical Hypotheses*, vol. 70, no. 1, pp. 182–185, 2008.

[23] D. Green, "Testing Randomness," *Teaching Mathematics and its Applications*, vol. 1, no. 3, pp. 95–100, 1982.

[24] A. Rapoport and D. Budescu, "Generation of random series in two-person strictl competitive games," *Journal of Experimental Psychology: General*, vol. 121, no. 3, pp. 352–363, 1992.

[25] K. Eliaz and A. Rubinstein, "Edgar Allen Poe's Riddle: Do Guessers Outperform Misleaders in a Repeated Matching Pennies Game?." http://arielrubinstein.tau.ac.il/papers/poe.pdf, Feb. 2008.

[26] L. von Ahn, "Games with a purpose," *Computer*, vol. 39, pp. 92–94, June 2006.

[27] B. Barak and S. Halevi, "A model and architecture for pseudo-random generation and applications to /dev/random," *ACM Conf. Computer and Comm. Security*, pp. 203Ű–212, 2005.

[28] J. von Neumann, "Various techniques used in connection with random digits," *Applied Math Series*, vol. 12, pp. 36–38, 1951.

[29] R. Shaltiel, "Recent developments in explicit constructions of extractors," *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, 2004.

[30] B. Barak, R. Shaltiel, and E. Tromer, "True random number generators secure in a changing environment." Cryptographic Hardware and Embedded Systems -

proc. CHES 2003, LNCS 2779, 166-180, Springer, 2003.

[31] P. Ayton and R. Falk, "Subjective Randomness in Hide-and-Seek Games," *Book of Abstracts of the 15th Bi-annual Conference on Subjective Probability, Utility and Decision-Making*, p. 37, 1995.

[32] A. Rubinstein, A. Tversky, and D. Heller, "Naive strategies in competitive games," *Understanding Strategic Interaction - Essays in Honor of Reinhard Selten, W.Guth et al. (editors)*, pp. 394–402, 1996.

[33] P. Miltersen and T. Sørensen, "A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament," *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007.

[34] J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P. Lu, and S. Sutphen, "Checkers Is Solved," *Science*, vol. 317, no. 5844, p. 1518, 2007.

[35] G. Costikyan, "I have no words and i must design." Interactive Fantasy vol. 2 (1994), available at http://www.costik.com/nowords.html.

[36] Y. Guiard and M. Beaudouin-Lafon, "Fitts' law 50 years later: applications and contributions from human–computer interaction," *International Journal of Human-Computer Studies*, vol. 61, no. 6, pp. 747–750, 2004.

[37] R. corp., "A million random digits with 100,000 normal deviates." http://www.rand.org/pubs/monograph_reports/MR1418/ (Accessed October 2008), 1955.

[38] G. Marsaglia, "DIEHARD: A Battery of Tests of Randomness," *See http://statf.su.edu/∼geo/diehard.html*, 1996.

[39] L. von Ahn and L. Dabbish, "Labeling images with a computer game," in *CHI* (E. Dykstra-Erickson and M. Tscheligi, eds.), pp. 319–326, ACM, 2004.

[40] S. Zhai, J. Kong, and X. Ren, "Speed-accuracy tradeoff in fitts' law tasks: on the equivalency of actual and nominal pointing precision," *Int. J. Hum.-Comput. Stud.*, vol. 61, no. 6, pp. 823–856, 2004.