Games for Extracting Randomness

Two computer scientists have created a video game about mice and elephants that can make computer encryption properly secure—as long as you play it randomly.

By Ran Halprin and Moni Naor

DOI: 10.1145/1869086.1869101

People have been sending messages in code almost since writing began—and others have been trying to crack those codes. The study of the design and breaking of codes is known as cryptography. With the advent of computers, the scope of cryptography has expanded significantly to now deal with methods for ensuring the secrecy, integrity, and functionality of computer and communication systems in face of an adversary.

Randomness is essential for addressing the main problems of cryptography (encryption, identification and authentication). When designing a code, one has to select its architecture—the mathematical method behind it. It is commonly assumed in cryptography that your adversary will know or find out the architecture of your code. The assumption is referred to as Kerckhoffs' principle. The part assumed to be unknown by the adversary, at least initially, is called the key. It should be selected from a large space of possible keys.

THE PROBLEM WITH RANDOMNESS

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. The most common solution is to find a source assumed to be somewhat random (often denoted "entropy source"), or at least unpredictable from the point of view of an adversary, and then apply some hash (mixing) function to it. The 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 or movement). This process is called randomness extraction.

Entropy is a measure of the uncertainty of a source or random variable. The most well known variant, the Shannon entropy, measures the expected length of the best encoding. For the purposes of randomness extraction, especially for cryptography, a more important measure is the min-entropy, the measure the probability of guessing the source, that is, the most common value. For a random variable X with n possible values the Shannon entropy is: $H(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$. The min-entropy is $H_\infty(X) = \min_{x\in\{0,1\}^n} \log_2 \frac{1}{\Pr[X=x]}$. It is known that $0 \le H_\infty(X) \le H(X) \le \log_2 n$

While in principle random extraction is a good approach, its implementation is often problematic and not based on a rigorous approach. There have been some well known failures of this type. Early in the history of the web, Goldberg and Wagner attacked Netscape's implementation of the SSL protocol—the cryptography designed to protect such things as secure internet credit card transactions. The attack was based on the Randomness Generator's entropy sources and refresh rate **[10]**.

Even more than a decade later, security vulnerabilities are still found in pseudo-random generators (PRG). For example, a bug in the pseudo-random generator of the Debian Linux operating system (including the popular Ubuntu Linux distribution) caused the generated cryptographic keys to be insecure, since they accidentally relied on very little entropy. This bug was only discovered in May 2008, two years after being introduced into the system **[20]**.

Entropy collection is the first critical step in generating pseudo-randomness. Today, randomness generators use one

or more of three types of entropy collection. One is background gathering—constantly collecting data from the operating system such as changes in memory or mouse position. This method is used for example by Windows and Linux. One problem here is that usage patterns might be predictable. The mouse might be frequently in the lower-left corner of the screen, for example, which makes it non-random.

An alternative is hardware gathering. The system observes external chaotic systems such as cloud patterns, electrical circuits, atmospheric noise or even lava lamps **[3, 11, 14]**. Such systems have large overheads, are physically complex and are therefore expensive to create and maintain. The third system is 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. 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 will be bored with requests for entropy and might perform very similar patterns every time they are prompted.

The issue of obtaining good randomness is not one of "send and forget." We do not want the system to have the property that a single leak compromises the system forever. For instance, HTTP session IDs are assumed to have randomness and should be hard to guess. Therefore we need methods to refresh our randomness periodically.

HUMANS AND RANDOMNESS

It is generally accepted that sequences and numbers generated by humans are far from being truly random. Ask someone to choose a number between 1 and 10 and they will usually choose 7. For the range 1 to 20, they usually choose 17. Other human biases in randomness assessment are well researched. There is the "hot hand," the tendency to believe that a winning streak will usually continue. Its inverse is the "gambler's fallacy," the belief that after a losing streak, the next attempt is more likely to be a success, and the related "flip bias," which is the all-too-human tendency to believe that a 0 is likely to be followed by 1, or, in roulette, that red will be followed by black, and vice versa. These beliefs have been shown, statistically, to be fallacies) **[4, 6, 18]**.

"In a random binary string of length 100 either the string 111111 or the string 000000 should appear with probability 95 percent, but a human finds that hard to believe."

"Flip bias" has been shown to exist in randomness generation as well **[16]**. Figurska *et al*. showed that humans asked to generate a string of digits tend to choose successive identical digits with probability 7.58 percent instead of 10 percent **[7]**. Thus they would produce sequences such as 5, 7, 9, 9, 4, 2 less frequently than they should. Due to flip bias, humans expect (and generate) sequences that contain shorter repeating sequences than true randomness would. For example, in a random binary string of length 100 either the string 111111 or the string 000000 should appear with probability 95 percent, but a human finds that hard to believe and would assess strings that contain them as non-random.

An interesting twist in the study of humans and randomness emerged when Budescu and Rapoport demonstrated that under some conditions humans can attain near-randomness **[15]**. It happens when they are playing the zero sum competitive game "matching pennies." In this game, each player chooses a red or a black card. The first player wins if each player chooses a different card and the second player wins if the cards are identical. When playing this game over many rounds, the players have an incentive to select cards randomly, so that their opponent cannot predict their choice—and human choices in matching pennies are much closer to random than when they actually try to simulate a random process.

Newer experiments with monkeys and bananas, and humans and other rewards, seem to support the claim that higher primates are able to behave almost randomly while participating in competitive games **[2, 8, 9]**.

IT IS ONLY HUMAN TO BE BIASED

While better than expected, human's gameplay patterns are still not perfectly random. Eliaz and Rubinstein produced a slight variant on matching pennies, in which the first player is called the "misleader" or "hider," whose specific task is to choose a card that would fool his opponent. The opponent is called the "guesser" or "seeker," who has to deduce from past behavior the card that the opponent has chosen.

Mathematically, the game is identical to matching pennies. The only difference is psychological. In matching pennies, the players have exactly identical roles. In the variant, one has become a hider, the other a seeker. Yet that change in psychology seems to affect the outcome. A guesser seems to have a slight empirical advantage over a misleader **[5]**. Specifically, guessers have a winning rate of up to 0.542 (rather than the expected 0.5).

The authors attribute this advantage mostly to the natural thought pattern used by guessers (trying to think like their opponent rather than think the opposite from their opponent), and their strategic framing (they are ordered to think where the other player hides, while the misleaders are not told to think where they will be looked for). They were able to eliminate this bias by using a variation of the game in which both players turned into guessers. (Player 1 chooses 'a' or 'i', player 2 chooses 's' or 't'. Player 1 wins if the word created is "as" or "it," player 2 wins if the word is "is" or "at"). This time, symmetry returned and the long-term results were 50 percent for both players.

A more disturbing bias in the context of our work is the tendency of winning guessers to repeat their choice (60.7 percent) and the tendency of winning misleaders to change their choice after success (57.7 percent). It is not clear how much this effect reproduces in more complex (non-binary) games.

A NEW TYPE OF ENTROPY SOURCE

Knowing that humans posses the capacity to generate randomness (at least to some extent) when they are playing some games, we can predict how random they will be. In order to extract entropy—that is, randomness—from human gameplay, we would like a game that encourages humans to play as randomly as possible.

Game theory often discusses games with mixed strategy, but most recreational games actually played by people have 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 poker) minimize the risk of losing. This includes also single-player video games such as *Pac-Man* or *Super Mario Bros.* These games rely on the player's skill in finding and performing winning sequences. Such games are obviously less useful for our entropy extraction, as play is usually far from random (although some entropy can be found in probably any gameplay sequence), so we want to identify games that encourage humans to be as random as possible.

The game matching pennies has two drawbacks. The first is that while it encourages humans to be random, it only generates one raw bit of data (which means at most one bit of entropy) per round, and a modern random key is at least 128 bits long.

The other problem is that we need a game that is at least somewhat interesting so that it entertains players long enough for them to generate sufficient randomness. Matching pennies is a rather boring game (Rapoport and Budescu had to pay their student test subjects to play it).

Another issue we need to tackle is what to do with the fact that people's choices are still not perfectly random and may in fact be quite far from random. These biases can be fixed by using randomness extractors (to be defined later) and the only assumption we need is that no single pattern is too dominant. That is, the probability that the adversary can guess perfectly all the opponent's moves is bounded by some small value—an assumption on the min-entropy of the player's actions.

"Some players use the natural tactic of repeating the same moves over and over from round to round [which prevents true randomness]."

OF MICE AND ELEPHANTS

We chose to study a natural extension of matching pennies called hide-and-seek. In the classic version of hide-and-seek, one player (the hider) chooses a cell on a vector or grid. 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.

Thus unadorned, the hide-and-seek game does not seem to be much more entertaining than matching pennies. To make the game more enjoyable so that people 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. An alternative might have been to develop the game of Battleship, with "fleets" secretly positioned on squared paper, and each fleet being shot at in turn by the opponent. As far as we know, no one has ever had to pay nine-year-olds to play Battleship.

Mice and elephants, however, is played on-screen, on a 512×256 pixel board which fits the screen nicely and produces 9+8=17 raw bits of entropy per user click. (512=29, 256=28). See **Figure 1**. The human player controls mice (1 in the first round, 2 in the second and third, 3 in the fourth and fifth, etc.), and the computer controls the elephant (the game is available online at http://math166-pc.weizmann.ac.il). In each round, the hider positions the mice on the grass, and the seeker positions the elephant. After both players perform their moves, 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 and flattens it. 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 to make the game more interesting. The increasing probability for losing in each round helps build tension in the game, especially when the player approaches his previous highest round. 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 his behavior.

Players cannot position the mice on top of each other, and the game field also has obstacles, which are positioned in popular click locations to prevent repetitive behavior.

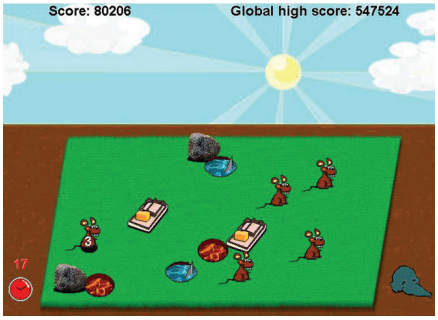
One problem we've seen is some players use the natural tactic of repeating the same moves over and over from round to round. A simple solution was having the elephant, with high probability, choose its moves uniformly from the player's recent history of moves (once the player builds such a history—before that the moves are always random). This ensures that playing a certain region too often will cause the player to be caught faster.

The game was released to the public as an experiment. The purpose was to study how people played it, and specifically how predictable (and therefore not random) they are. 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 the elephant and survive as many rounds as possible. For analysis of the results of this experiment, see the full publication **[12]**.

SECURE PSEUDO-RANDOM GENERATION

To fully specify the system, we need to describe how we obtain randomness

Figure 1: Positioning mice in the game mice and elephants generates randomness. (Graphic courtesy of Anat Iloni).



from the gameplay. We construct a fully functional PRG system which can be tested online http://math166-pc.weizmann.ac.il or downloaded from http://www.neko.co.il/MAE-offline.html. The system lets the user play and collects the gameplay data. This data is not perfectly random, but should have sufficient entropy. Using a randomness extractor we create from this data an almost-random sequence, which we then use to seed a cryptographic pseudo-random generator.

RANDOMNESS EXTRACTORS

What does it mean to be far from uniform or close to uniform? For this we use the measure known as statistical distance: Two distributions P and Q over the same domain T are **statistically ε-close** if: $\frac{1}{2}\sum_{x\in T}|P(x)-Q(x)|\leq\varepsilon$.

The goal of randomness extractors is to start with a source that is far from uniform and transform it into one that is close to uniform. The earliest discussion of using "weak" random sources in order to get full randomness was made by von Neumann, who considered the case of a biased source that outputs a 0 with probability p and 1 with probability $(1–p)$ **[19]**. Uniform randomness can be extracted from this source by considering pairs of bits: If the pair is 01 we output 0. If the pair is 10 we output 1. If the two bits are identical, we ignore this pair. The resulting sequence is perfectly uniform, regardless of the value of p.

In recent times the theory of randomness extractors has been thoroughly investigated. Efficient methods have been found to extract randomness from sources without a known structure, where the only requirement is high min-entropy and some random "seed."

A (k, ε)-extractor is a function Ext: $\{0,1\}^n$ x $\{0,1\}^d$ — $\{0,1\}^m$ such that for every distribution X on $\{0,1\}^n$ with $H\infty(X) \geq k$, the distribution $Ext(X, s)$ (where $s\,\varepsilon_R\,\{0,1\}^d$, 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 extractor 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 the seed is provided, we call the extractor "strong."

A (k, ε)-strong extractor is a function Ext as above, where the distribution $Ext(X, s)$ followed by s is statistically ε-close to the uniform distribution on $\{0,1\}^{m+d}$. Using a strong extractor implies that the seed is reusable. In many applications, it is therefore possible for this seed to be hard-wired into the system once in its creation. There are good constructions of strong extractors, and a fairly simple but quite powerful one can be based on random linear functions over finite fields. For more about extractors see Shaltiel's survey **[17]**.

CRYPTOGRAPHIC PSEUDO-RANDOM GENERATORS

A cryptographic pseudo-random generator is an efficiently computable function that, given a random seed, creates a longer random looking sequence of bits. This sequence should look indistinguishable from random for any observer who computation power is limited. In general, for every feasible test we want the probability that the output of the generator will "pass" the test to be close to the probability that a truly random string passes the test. The precise definition of Cryptographic PRGs is beyond the scope of this paper (for more details see, for example, the textbook by Katz and Lindell **[13]**.

"Under some conditions humans can attain near-randomness. It happens when they are playing the zero sum competitive game 'matching pennies.'"

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 sequence could be used to verify these guesses.

In order to address this problem, Barak and Halevi introduced a robust pseudo-random generator **[1]**. A robust PRG has two input functions: *refresh()* that refreshes its entropy, and an output function *next()* that returns a piece of the pseudo-random sequence. A robust PRG should satisfy 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 generated prior to the break-in, should still be indistinguishable from random to the adversary (naturally, this means the past states should not be reconstructable).

Backward security (break-in recovery): assume the adversary learns the current internal state. 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 major advantage of robust PRGs is the ability to combine independent

sources of entropy for a proven increase in security. This possibility can also be seen as a motivation for the current work and for finding other sources of entropy as different as possible from those in use today.

Barak and Halevi show how to construct a provably robust PRG from any cryptographically secure PRG.

RANDOMNESS THROUGH FUN

Using the results from the experiment, we estimated how much randomness people were putting in this game. Based on the estimation of the min-entropy, we found that 39 mouse clicks, are sufficient to seed a PRG with 128 bits which are 2128-close to random. That is, assuming the game was played with enough randomness, the resulting string has high min-entropy. The results of extraction are inserted as a seed into an implementation of the Barak-Halevi system that we created, which can be reseeded by gameplay whenever the user wishes.

How do we know whether the result is indeed random? The disappointing answer is that we cannot really know. It is very difficult to assert that the results are indeed random due to the universal requirement of cryptographic pseudo-randomness—that it fools all possible tests. Therefore, one possibility for a sanity check is to use a standard test of randomness. One such randomness assessment suite is known a DIEHARD. In order to test our system, we let it generate millions of keys (with the clicks played in the game as a single randomness source) and ran them through the DIEHARD suite, which gave it very good results, similar to Microsoft Windows' Cryptographic Randomness Generator.

Our main conclusion from this work is that using human gameplay as a randomness source is a valid option for generating secure randomness, whether by itself when other forms of randomness are not available, or by combining it with other entropy sources. We also learned of the power of games as a motivator to participate in experiments, as we collected more information than is customary in such research and more easily. And—always useful given research budgets—we did not have to pay our participants.

"Using human gameplay as a randomness source is a valid option for generating secure randomness"

The downside is that we don't really know who the group of participants was, their demographics, and how accustomed they are to playing computer games.

Biographies

Ran Halprin holds a MSc. in computer science from the Weizmann Institute of Science. His research interest is in human computation and its applications, mainly in cryptography.

Moni Naor is a professor at the Department of Computer Science and Applied Mathematics at the Weizmann Institute of Science and the incumbent of the Judith Kleeman Professorial Chair. His research interest is in foundations of computer science, especially cryptography.

References

1. Barak, B. and Halevi, S. 2005. A model and architecture for pseudo-random generation and applications to /dev/random. *ACM Conf. Computer and Comm. Security*, 203˝U–212.
2. Barraclough, D., Conroy, M., and Lee, D. 2004. Prefrontal cortex and decision making in a mixed-strategy game. *Nature Neuroscience 7*, 4, 404–410.
3. Bernstein, G. and Lieberman, M. 1990. Secure random number generation using chaotic circuits. Circuits and Systems, *IEEE Transactions on 37*, 9, 1157–1164.
4. Burns, B. and Corpus, B. 2004. Randomness and induction from streaks: "gambler's fallacy" versus "hot hand". *Psychonomic Bulletin and Review 11*, 179–184.
5. Eliaz, K. and Rubinstein, A. 2008. Edgar Allen Poe's Riddle: Do Guessers Outperform Misleaders in a Repeated Matching Pennies Game? http://arielrubinstein.tau.ac.il/papers/poe.pdf.
6. Falk, R. and Konold, C. 1997. Making sense of randomness: Implicit encoding as a basis for judgment. *Psychological Review 104*, 2 (April), 301–318.
7. Figurska, M., Stańczyk, M., and Kulesza, K. 2008. Humans cannot consciously generate random numbers sequences: Polemic study. *Medical Hypotheses 70*, 1, 182–185.
8. Flood, M., Lendenmann, K., and Rapoport, A. 1983. 2×2 Games played by rats: different delays of reinforcement as payoffs. *Behav Sci 28*, 65–78.
9. Glimcher, P., Dorris, M., and Bayer, H. 2005. Physiological utility theory and the neuroeconomics of choice. *Games and Economic Behavior 52*, 2, 213–256.
10. Goldberg, I. and Wagner, D. 1996. Randomness and the netscape browser. *Dr. Dobb's Journal*, 66–70.
11. Haahr, M. Random.org: true random number service. http://www.random.org.
12. Halprin, R. and Naor, M. 2009. Games for extracting randomness. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, article 12, California.
13. Katz, J. and Lindell, Y. 2008. *Introduction to modern cryptography*. Chapman & Hall/CRC.
14. Noll, L., Mende, R., Sisodiya, S., et al. 1998. Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system. US Patent 5,732,138.
15. Rapoport, A. and Budescu, D. 1992. Generation of random series in two-person strictl competitive games. *Journal of Experimental Psychology: General 121*, 3, 352–363.
16. Rapoport, A. and Budescu, D. 1997. Randomization in individual choice behavior. *Psychological Review 104*, 1, 603–617.
17. Shaltiel, R. 2004. Recent developments in explicit constructions of extractors. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*.
18. Tversky, A. and Gilovich, T. 2004. The "Hot Hand": Statistical Reality or Cognitive Illusion? Preference, Belief, and Similarity: Selected Writings.
19. von Neumann, J. 1951. Various techniques used in connection with random digits. *Applied Math Series 12*, 36–38.
20. Weimer, F. New openssl packages fix predictable random number generator. http://lists.debian.org/debian-securityannounce/2008/msg00152.html.