Trust, and public entropy: a unicorn hunt

Arjen K. Lenstra and Benjamin Wesolowski

EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland

Abstract. Many applications require trustworthy generation of public random numbers. After presenting a number of these applications, and reviewing various techniques aiming at providing incorruptible public randomness, we discuss the design of a secure, trustworthy random beacon. We derive what constraints are unavoidable to reach certain security guarantees.
Keywords: public random number generation, random beacon, trust, (un)trusted third-party.

1 Introduction

Sources of randomness are critical components in cryptography. They are often supposed to operate in a concealed, private environment, generating bits that are meant to be kept secret. There are however applications that require a form of publicly available randomness that cannot be predicted or manipulated. Obvious examples are national lotteries and sporting events draws. It also plays a role in governance, for sampling of assemblies or citizen juries, or tie-breaking in elections. Trustworthy public sources of randomness also appear in more technical settings. Rabin introduced in [26] the concept of a *random beacon* – an online service that publishes allegedly unpredictable numbers at regular intervals – to provide trust between communicating parties in various cryptographic applications. Besides its applications in cryptographic protocols, public randomness may also be used in the design of cryptographic standards. The seeding of the generation of standardized parameter choices for elliptic curves has been a matter of particular concern [4].

There exist today a few services that continuously make available fresh random numbers, such as [24] and [27]. In these systems, no mechanism allows a user to verify the freshness or correct generation of the published numbers, so these services can only be considered as trusted third-parties. These last years have seen various attempts at developing and analysing methods to build trustworthy public sources of randomness. Clark and Hengartner proposed in [12] a beacon based on stock market prices. The security thus relies on the presumed unmalleability of published financial data – hardly sufficient to convince even modestly skeptical users. Relying on a number of external, presumably independent, sources of randomness is also the approach taken by Baignères et al. in [2]. They describe how to combine the outcome of several national lotteries to generate random public parameters for elliptic curve cryptography. Their work relies on the idea that these lotteries are difficult to manipulate. In an attempt to get rid of all assumptions on the incorruptibility of external sources, in [19] we constructed and proved the security of a verifiable beacon that anyone can influence, but no one can bias or predict, by feeding public input data to a slow hash function. It has also been suggested to exploit the inherent unpredictability of blockchains, in particular that of the Bitcoin protocol [22], to build a decentralised random beacon. This idea is analysed by Bonneau, Clark and Goldfeder in [6], and later by Pierrot and Wesolowski in [25]. A source of randomness internal to Bitcoin allows that randomness to be used by the currency's scripting language, considerably extending the range of implementable smart contracts.

Some applications of public randomness are described in Section 2. Various methods to provide such randomness in a trustworthy manner are discussed in Section 3. In Section 4, we discuss possible design choices for the construction of a secure, unpredictable source of public randomness, and derive what constraints are unavoidable to reach certain security guarantees. Parts of the descriptions below are taken from our earlier [19].

2 Applications of public randomness

2.1 Protection by random beacons

Considering transaction protocols that were proven to be impossible without help from an intermediary, Rabin proposed in [26] to implement them in a simple and efficient way using a very rudimentary form of third-party: a *beacon* emitting random integers at regular intervals. He showed how to instantiate fair contract signing, disclosures of confidential information, and certified mail in an electronic mail system. It is worth noticing that since then, other protocols for these problems have been developed without need for a third party, satisfying weaker forms of security than what was proven impossible without a third party. These applications are further explained below.

Assume two parties A and B negotiate a contract c. Once they agree on c, they both want the other party's signature on c. If A signs c and sends it to B, then B finds himself in a position of strength as A is already committed to the deal while B is not. This unfair situation would be avoided in a *fair* contract signing protocol. A *disclosure protocol* deals with the situation where A and B have to agree that A will disclose some confidential piece of information s to B, and B promises not to divulge it. When A then reveals s to B, a receipt from B will be required to enforce the agreement, and the protocol should prevent unfair situations where B obtains s but leaves A without a receipt. *Mail certification* allows A to send a message m to B and obtain a receipt. Here, m need not be confidential, and B might be allowed to inspect its content before accepting the transaction or not. If B accepts, both parties should be able to prove that the transaction took place.

In the proposed implementations of the above applications the role of the beacon is limited to broadcasting the random numbers. Therefore the third-party that is responsible for the beacon does not have to perform any protocol-specific computation, making a beacon a very versatile and scalable solution. The security of the protocols relies on the unpredictability of the generated numbers: a party able to predict future values of the beacon could cheat. The fact that only one beacon is needed for different applications that may run simultaneously, and that it has a very simple behaviour, seems to make it easy to build a beacon, but the trust issue has to be addressed. Rabin concludes that the beacon has to be a trusted party. Section 3 will go through possible solutions to build a beacon that need not be trusted a priori. Another potential issue is that the random beacon values may be used for purposes for which they are not intended, such as seeding a pseudo-random number generator – with potentially disastrous consequences if secrecy of the resulting pseudo-random numbers is assumed.

2.2 Seeding cryptographic standards

Seeding the generation of elliptic curves. Another application of uncontestable generation of random numbers is the seeding of the generation of standardized parameter choices for elliptic curve cryptography. Although there are many elliptic curve parameters that would be

suitable for cryptographic applications, there is only a small set of elliptic curve parameters that are recommended or standardized for general use [5]. Using either one of these curves implies trusting the way it was generated. A particular choice of parameters could hide special properties and potential weaknesses known only to the party publishing the curve: [4] elaborates why this could be problematic even if care seems to be taken to avoid trust issues. Having users use their own personalized parameters is not an option yet: due to the current state of the art in elliptic curve point counting, generating good random elliptic curve parameters is a tedious process whereas parameters that can be quickly generated (using the complex multiplication method) are frowned upon – albeit for unknown reasons.

The standardization effort has seen different strategies emerge to justify the choice of any particular set of parameters. The usual strategy to find a suitable elliptic curve is to define a set of requirements (addressing security or efficiency), and a procedure to derive a sequence of elliptic curves from an input seed (the *sampling* procedure). The first elliptic curve in the sequence that fulfills the set of requirements is chosen. See [16] for instance for more details on the design of such a procedure. The trust issue boils down to the choice of the seed.

In most cases, the procedure and the seed are defined simultaneously: the seed is not chosen randomly but is specified together with the other parameters. As argued in [4] , these techniques result in malleable parameters, because in order to find a curve with some targeted properties the generating party has the freedom to play with the set of requirements, the sampling procedure and the seed. For instance, the seeds for the NIST P-curves [23] are hashes of strings. In this case, the resistance to manipulation is meant to arise from the pre-image resistance of the underlying hash function, rather than from a trustworthy choice of seed. The seeds for the Brainpool curves [21] are digits of natural constants (namely, of $e = \exp(1)$ and $\pi \approx 3.14$). So-called *rigid* curves, such as the NUMS curves [8] or Curve25519 [3], justify the choices made by arguing that the entire generation process is carried out in a canonical or optimal way. This has also been shown to leave a number of degrees of freedom.

Another approach consists in defining the procedure ahead of time, in a completely public and unambiguous way, and later generate an independant, trustworthy random number, which is then provided as a seed to the procedure to produce a curve. This is the approach taken by *trx* [20]: an online service that continuously produces new elliptic curves suitable for cryptographic applications, based on incorruptible random numbers generated using the *unicorn* protocol [19]. The Million dollar curve [2] has been generated in a similar fashion, but using entropy from national lotteries around the world.

Seeding other cryptographic standards. In the same vein, public randomness could also be used to generate constants for other kinds of cryptographic parameters. In [28], it is described how to design the constants of the S-boxes in some block ciphers to hide a trapdoor. Therefore the choices of these constants need to be carefully justified. Similarly, [1] exposes a way to weaken SHA-1 and find collisions by simply tweaking its round constants. To establish public trust in standardized block ciphers and hash functions, it must be possible to prove that their parameters have been chosen in a trustworthy manner.

2.3 A tool for democracy

Randomness can play a crucial role in various models of governance. The first known democracy in the world, in the Greek city-state of Athens, distributed the power to assemblies of randomly selected citizens. In today's world, the benefits of sortition-based democracy are defended by some as a fairer alternative to elected assemblies. Without going as far as a full

Athenian-like democracy, more familiar modern-day models of governance can make use of random-sample voting: instead of consulting the full population for elections or referenda, one can randomly select a small, yet statistically significant, sample of voters. Such a system is advocated as leading to a better quality voting at a far lower cost [10]. In Switzerland, the population regularly votes for diverse elections, referenda, and popular initiatives. For reasons of cost, these so-called *votations* are organized four times per year, each time about multiple topics. Being questioned so often about so many issues sometimes far from their everyday life, voters can feel overwhelmed and unable to develop an informed opinion for each question. In such a system, replacing the full population by random samples would have multiple advantages: while remaining statistically representative as long as the sample has an appropriate size, the costs would be dramatically reduced. At the same time, each voter would be requested much more rarely and about a single topic at a time, allowing for a more important involvement.

Secure random sampling. Whether it be for legislative assemblies, small samples of voters, or juries for public policy, the random sampling must be conducted in a trustworthy, incorruptible and verifiable manner. Designing such a procedure is a delicate task which [13] tries to address. It boils down to two main components: a public random number generator, and a deterministic procedure which, when fed an input number, outputs a sample of citizens in a completely unambiguous and verifiable manner. This second component needs to be precisely defined and published ahead of the random number generation. An example of such a procedure can be found in [13, Section 4]. See [10] for discussions on how to design this component in order to render impractical vote buying or other ways to influence voters.

The first component is critical to the fairness of the outcome: it must provide convincing evidence that the random number was not cooked to result in a biased sample of voters. The author of [13] suggests to select different sources of randomness in advance (examples include government run lotteries, the daily balance in the US Treasury, or sporting events), and to combine the outcomes via a secure hash function. Various strategies are conceivable to manipulate those sources. Great care needs to be taken to audit each of them, and even if various presumably independent parties are involved, not every skeptical citizen can be given the chance to personally make sure everything went right and no form of manipulation happened. It would be immensely preferable to rely on an actual trustworthy, secure source or public randomness.

Cryptographic elections. Public randomness has also found applications in the context of cryptographic election systems: auditing via random selection and generation of random challenges (often used to establish that the tally was computed correctly) both require an unpredictable and incorruptible source of randomness. In at least two cases ([14] and [11]) the random generation was based on financial data.

2.4 Smart contracts and crypto-currencies

Randomness for smart contracts. Verifiable public randomness would be a powerful addition to the smart contracts' toolbox. This concept of self-enforcing contracts turned out to be especially useful in the context of crypto-currencies, such as Bitcoin [22] or Ethereum [33]. These systems have scripting languages which can be used to define specific conditions under which funds can be moved. To include some randomness in smart contracts, a verifiable, secure source of randomness would be necessary. At the moment even the currencies with the most powerful scripting languages do not allow random execution. A few Bitcoin pro-

tocols are however already relying on the blockchain (Bitcoin's data structure) as a source of public randomness to design secure multi-player lotteries, non-interactive cut-and-choose protocols [6], or randomized mixing fees [7]. A cut-and-choose protocol deals with the situation where a party tries to convince another party that some data he sent was honestly constructed according to an agreed-upon method. The beacon-based method proposed in [6] implements the first self-enforcing, non-interactive cut-and-choose scheme.

Preventing selfish mining. Bitcoin's blockchain is created by a set of active participants, the *miners*, who in exchange for their work can receive a reward consisting of a few bitcoins. More details about blockchains and miners are provided in Subsection 3.4. It is possible for large groups of miners to maliciously boost their rewards via a strategy called *selfish mining* [15]. For selfish mining to be profitable, the group should control at least 25% of the worldwide mining power. In [18], a modification of the Bitcoin protocol is described that increases this threshold: assuming the existence of an unpredictable public source of random numbers – which can be used to create unforgeable time-stamps –, selfish mining in this modified protocol is profitable if and only if the group controls at least 32% of the worldwide mining power. Selfish mining involves withholding valid blocks found by a miner. When included in a block, the timestamp guarantees that it was not mined *before* a certain point in time, and it can be detected if a miner has been withholding that block for a prolonged period of time.

3 Generating public randomness

3.1 Using widely accessible real-world entropy

A first, straightforward way to generate some public random numbers would be to commit in advance to using the result of some future, publicly verifiable observation that is expected to be impossible to manipulate. This is for instance the approach taken in [12], where a beacon is described that derives random numbers from the daily closing prices of a number of common stocks. Financial data has also been used as a source of randomness in [14] and [11]. It is easy to imagine that financial exchanges could subtly adjust the prices they announce to bias the "random" output.

The idea of using the results of national lotteries has also been raised [2]. Complex transparent machines with balls flying around in seemingly total chaos, as commonly used for national lotteries, are not incorruptible [31,9], and the situation gets even worse when the winning numbers are generated by a computer [30]. The process described in [2] combines the outcomes of multiple, seemingly independent lotteries – a practical difficulty which they can deal with because they only need to generate *one* number, with no strong calendar constraint. It raises a security issue: it is sufficient to corrupt the chronologically last lottery draw to significantly manipulate the outcome. This is referred to as the *Last Draw Attack*. The proposed countermeasures are either to only use simultaneous lottery draws (but it is unlikely that one could get independent lottery operators to synchronise their draws), to rely on a third party that is trusted to be independent from that last lottery (but if two parties are trusted to be fully independent, the whole trust issue is easy to solve), limiting the entropy of the last draws (the potential weakness of the very last draw is diluted among the last few draws), or to combine the different lotteries in a secure way that prevents manipulations, using slow functions [19], which is the core of the protocol described in Subsection 3.3.

Many other forms of publicly available entropy are conceivable: sports results, meteorological measures, or pictures of sunspots. The issues raised are always the same: to what extent are they susceptible to manipulation, and how can they be securely combined.

3.2 The NIST random beacon

In reaction to the number of potential applications, NIST has implemented their own source of public randomness [24]. This beacon outputs every minute a string of 512 random bits. Extensive documentation is provided that explains how these "truly random" bits – almost in a metaphysical sense – are generated based on quantum mechanical phenomena.

This beacon is advertised to provide *unpredictability*. Even though the machine that is described can be argued to be perfectly unpredictable even by the party running it, users of the beacon have no formal guarantee whatsoever that NIST is truly complying with the procedure. The bits as produced by the beacon come with no proof that they have been generated by the quantum machine as described.

3.3 *Unicorn*: uncontestable random numbers

Unicorn, as proposed in [19], provides a way to create incorruptible random numbers, the correct generation of which can be verified by the most mistrustful users. It relies on a simple observation: a number can be fully determined at point in time t, while none of its bits can actually be known to anyone before time $t + \Delta$ when its computation is completed, where $\Delta > 0$ is some delay. *Unicorn* makes use of a delaying function called *sloth* (for *slow-timed hash*): a hash function that takes, for any pre-specified value $\Delta > 0$, at least wall-clock time Δ to compute, irrespective of the amount of parallel computational power available, and whose outcome, once computed, can quickly be verified by anyone. The design of such a function takes inspiration from the literature of time-sensitive cryptography, introduced by Rivest, Shamir and Wagner in [29].

Let C be a central party running *unicorn*. A decentralised version is straightforward to conceive, but we chose to present it in a centralised context for clarity. Also, C need not be a trusted party, as all actions taken by C will be publicly verifiable. The protocol goes as follows. In a first phase, everyone is invited to contribute any type of home-made entropy. Any party who wants to get involved can then send its contribution to a public bulletin board or publish it on twitter. C collects the contributions received during the first phase, and concatenates them to form a public string s_0. Each contributing party can check that its contribution was included in s_0. Optionally (and on top of this), C can generate its own random contribution s_1 which is not published, but which is committed to. The second phase begins: the concatenation of s_0 and s_1 is fed to *sloth*, with the guarantee that the outcome cannot be known to anyone (including C) before a prescribed amount of time – say, ten minutes – has elapsed. At the end of the computation, the outcome, which is the verifiable random number, is published, alongside data for its verification, including an opening of the commitment on s_1. Note that the order of the concatenation of the contributions (to s_0) is significant, but because s_0 is made public right away while the resulting *sloth*-value cannot be known until ten minutes later, C cannot make an informed decision between different orderings.

Assuming the slowness of *sloth*, the protocol is proven secure in [19] in the worst case situation where any particular party, say user U, can only trust him or herself: the other

contributors to s_0, the parties gathering the data (e.g., twitter), generating s_1 and computing *sloth* might all be colluding to manipulate the outcome, while still trying to convince U that the outcome was honestly generated. Security in the proposed model renders any such manipulation infeasible: any manipulation of the outcome will be detected by a fast run of the public verification procedure.

It also allows to transfer trust: if U did not get the chance to take part in the protocol, security in the proposed model implies that U can still trust the outcome if U trusts at least *one* of the contributions included in s_0 (either because U has personal trust in the contributor, or because s_0 includes some piece of time-stamped information that U is ready to believe was not predictable). An interesting side effect is that U can also trust the outcome if U knows that two of the contributors are unlikely to have colluded, even if U does not trust either of them.

Note that it would be possible for the party C running *unicorn* to discard U's contribution to s_0, e.g., by claiming that it was sent after the deadline, maybe due to clock mis-synchronization. Despite the fact that a careful contributor would take this claim with high suspicion, this strategy does not allow manipulation because the attacker would need to discard *all* the honest contributions, thereby failing to provide trust to anyone suspicious about any other party not playing fair. Making a change to a contribution to s_0 (by the party running *unicorn*) would have the same effect.

Also, it is important to notice that the possibility to manipulate s_1 (the part of the seed generated by C) does not negatively affect the security in the proposed model as long as it is committed to in due time. This s_1 serves other purposes exposed in the original paper. As a final note, it is trivial to observe that the protocol does not provide more guarantee than the methods expounded in subsections 3.1 or 3.2, in the case where nobody contributes to s_0.

3.4 Bitcoin as a source of public randomness

In another attempt to build a decentralised beacon with no trusted parties, it has been suggested to exploit the inherent unpredictability of blockchains, in particular that of the Bitcoin protocol. A blockchain is a form of distributed database, introduced in [22] as the backbone of the Bitcoin protocol (and a classical cryptographic construction, cf. [17]). At regular time intervals, a new block is added to the blockchain, testifying for the latest transactions that happened in the Bitcoin worldwide system. To be added to the chain, a block needs to be *valid*, which implies in particular that its hash value for some chosen secure hash function has a certain number d of leading zeros. The process of finding a valid block is called mining, and it simply consists in hashing a block with slight modifications until a valid version is found; on average this requires 2^d attempts, and thus a substantial amount of time even for modest d-values (the current d-value in Bitcoin is approximately 64). This mining task is performed by *miners* in exchange for a reward (a few bitcoins) whenever they find a valid block.

Assuming the underlying hash function is secure, one can derive that each valid block found via the mining process contains at least d bits of computational min-entropy. The computational cost of the mining process seems to render this entropy difficult to predict or manipulate, and has in fact already been used as a source of public randomness, in particular in some Bitcoin lotteries. Such a lottery would go as follows: first the index of a future block of the chain is announced, the *decisive* block. Then, the lottery tickets are sold, but the sale must end before the decisive block is mined. Once the decisive block has been mined, its bits

are used as a random seed for a sampling function which picks the winner among the lottery tickets.

Bonneau, Clark and Goldfeder in [6] proposed a first security analysis of this technique. They consider a model where the attacker controls all the miners, and suffers a monetary penalty whenever a miner finds a valid block that the attacker does not want to see included in the blockchain. This is a bribing model, where the penalty is meant to compensate for the loss of the reward going to a miner who finds a valid block. They come to the conclusion that the adversary's expected cost is $2r$ to enforce a particular value of a random bit generated by this beacon, where r is the reward for one valid block. That is an expected cost of US$12,000 at the time of publication.

This analysis suffers a few limitations which makes it difficult to analyse the threat of this attack: it only considers financial costs, with no computational considerations, it assumes all the miners are willing to blindly accept the bribes, and it only considers attacks that force the value of the beacon rather than simply introducing a bias, which might be doable at a much lower cost. A subsequent analysis, in [25], overcomes these limitations. It allows for instance to conclude that, again when a single bit is generated, an adversary controlling only a quarter of the miners (rather than all of them as in the previous model) can bias the probability of obtaining a particular value from 0.5 to 0.6 for an expected cost of US$9,400. A cheaper strategy can increase the probability from 0.5 to 0.57 for only US$1,320; and on the other hand, with a higher cost it can increase up to 0.74. This shows that in their current form, blockchain-based random beacons are not a secure source of public randomness.

4 Discussion

These various directions in the design of a secure source of public randomness all come with a different set of assumptions, advantages and drawbacks. The current form of the NIST random beacon requires full trust in the single entity that built, runs, and audits the source. But an outsider cannot in any way ascertain that the random numbers have been generated as advertised. The idea of using lotteries, financial data or other forms of publicly available entropy attempts to address the lack of verifiability by combining a number of pre-specified, different, and presumably independent sources of entropy. This may increase the trust in the party running the beacon, but introduces a complication: that party cannot control the time of the day, or even the day of the week, when entropy can be harvested. It raises two issues: firstly, it makes it difficult to output a beacon value at a reasonably high frequency (e.g., every minute like the NIST beacon). Secondly, it is sufficient to corrupt the chronologically last source of entropy to manipulate the beacon value: once the values from all the other sources are publicly known, the last source fully determines the outcome.

Both *unicorn* and blockchain-based methods get rid of the assumption that there is a trusted external source of randomness, and replace it with computational assumptions. This is a necessary step to obtain some security in the worst case situation where a user of the beacon does not trust anyone but him or herself.

While the blockchain method is vulnerable to manipulation by a group of corrupted miners, the *unicorn* protocol is provably secure under the sole assumption that the underlying *sloth* function is slow to compute. Of the methods discussed here it is the only one that allows anyone to make personally sure that the generation process was done honestly. Two drawbacks however remain. These are discussed below, after which it is informally shown that both drawbacks are unavoidable.

The first drawback is that *unicorn* requires public input. Trust can only be provided to users who trust at least one of the contributions to the public seed s_0: because not every single user of the beacon can be expected to personally take part in the generation process *unicorn* cannot be expected to be provide *global* trust. On the bright side however, it is sufficient for a user to trust any single one of the contributions. Financial data, lottery draws, Bitcoin blocks or NIST-generated values can *all* be contributed to the s_0-value for an outcome of this beacon, and a user trusting any *one* of these contributions will be able to trust the outcome. Note the contrast with the idea from [2] of combining lottery draws and such: even if all the contributions are trusted except the last one, then the outcome cannot be fully trusted. This shows that assuming the slowness of *sloth*, a *unicorn*-based beacon is stronger than all the previous constructions thanks to the flexibility allowed by the openness of the contribution system.

The second drawback is the time delay imposed by *sloth*. Once the public input s_0 has been collected, the outcome will only be known at the end of the *sloth* computation, which is designed to take time. Even though this delay is key to the security of *unicorn*, and verifiable "freshness" intuitively looks like a necessary constraint to trust a beacon's outputs, one can wonder if this can be proved.

In the design of a beacon secure in the model where users do not trust anyone but themselves, it can informally be argued that both drawbacks noted above are unavoidable and thus that public contributions and a delay are necessary. First, suppose the beacon, which runs a protocol P, does not allow a user U to contribute his own entropy. In the worst case situation where U believes everyone is secretly colluding against him, the beacon has no way to convince U of the freshness of the output random number, as U might believe that the protocol P has secretly been run in advance any number of times required to allow the beacon to choose a desirable outcome[1]. So the beacon must accept input from the concerned users (this is s_0 in *unicorn*).

Secondly, suppose the beacon does not commit to the exact computations that will be done before publishing the output (this is done in *unicorn* by letting s_0 be publicly available immediately, and by publishing a cryptographic commitment on s_1 before starting the *sloth* computation). Without such a commitment, the beacon could, right after having received the public contributions s_0, run in parallel the protocol P for various values of $\tilde{s}_0$, each being a concatenation of all honest contributions to s_0 and a single unique malicious contribution. Once all the parallel computations are done, the beacon can choose its favorite outcome, and the associated variant $\tilde{s}_0$. The commitment defines a point in time at which the output is already determined – even though it may not yet be known: once the beacon has committed to the exact computations that will be performed, the outcome is the result of this computation and cannot be tampered with anymore. Now, suppose there is no delay between the commitment (which determines the output) and the moment the beacon is done computing the output (this is the delay imposed by the slowness of *sloth* in *unicorn*). It means the beacon already knows the output at the moment it is committing to it. This again allows the beacon to choose the variant $\tilde{s}_0$ that it prefers depending on the output it leads to. The delay forces the beacon to commit to some output *before* having got the time to actually compute it, and therefore cannot do any meaningful guess on the most favorable $\tilde{s}_0$.

[1] Attempts to justify that nothing was done in advance by involving sporting results or weather data, not only would provide no formal guarantee, but would definitely not help convincing a user U suffering from The Truman Show delusion [32].

5 Conclusion

The last four decades a lot of thought has been put in design and implementation of methods that can be used to protect information, allowing the widest possible interpretation of what it means to *protect information*. Relatively little effort has gone into building confidence among the user population that these methods have been incorporated in an effective manner in our current communication infrastructures. Quite on the contrary: any even half informed user has every reason to be highly skeptical.

Given the complexity of the hardware and software involved, there may be many entities – of any number of different nationalities – that seek to profit from undocumented features. Addressing this problem and offering users assurance that they can control and verify how their data are collected and processed is an unsolved problem for which we do not offer a solution. We wish we could. In the very limited scenario, however, where public random numbers are generated, we offer an approach that considerably improves on previous methods and that achieves our goal of giving the user precisely the right level of control.

Acknowledgement

The second author thanks the Swiss National Science Foundation for its support via grant number 200021-156420.

References

1. A. Albertini, J.-P. Aumasson, M. Eichlseder, F. Mendel, and M. Schläffer. Malicious hashing: Eve's variant of SHA-1. In A. Joux and A. Youssef, editors, *Selected Areas in Cryptography – SAC 2014*, volume 8781 of *Lecture Notes in Computer Science*, pages 1–19. Springer International Publishing, 2014.
2. T. Baignères, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain. Trap me if you can – million dollar curve. Cryptology ePrint Archive, Report 2015/1249, 2015. `http://eprint.iacr.org/`.
3. D. J. Bernstein. *Public Key Cryptography - PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings*, chapter Curve25519: New Diffie-Hellman Speed Records, pages 207–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
4. D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, T. Lange, R. Niederhagen, and C. van Vredendaal. How to manipulate curve standards: a white paper for the black hat. Cryptology ePrint Archive, Report 2014/571, 2014. `http://eprint.iacr.org/2014/571`.
5. D. J. Bernstein and T. Lange. Safecurves: choosing safe curves for elliptic-curve cryptography. `http://safecurves.cr.yp.to`, accessed 2 September 2014.
6. J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015, 2015. `http://eprint.iacr.org/2015/1015`.
7. J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security: 18th International Conference, FC 2014*, pages 486–504, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
8. J. W. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, pages 1–28, 2015.
9. Businesspundit. Biggest lottery scandals. `http://www.businesspundit.com/5-of-the-biggest-lottery-scandals`, 2012.
10. D. Chaum. Random-sample elections: Far lower cost, better quality and more democratic, 2012.
11. D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *USENIX/ACCURATE Electronic Voting Technology Workshop (EVT)*, 2008.
12. J. Clark and U. Hengartner. On the use of financial data as a random beacon. In *USENIX EVT/WOTE*. USENIX Association, 2010.

13. D. Eastlake 3rd. Publicly verifiable nominations committee (NomCom) random selection, 6 2004. RFC 3797.
14. A. Essex, J. Clark, R. T. Carback, and S. Popoveniuc. Punchscan in practice: an E2E election case study. In *IAVoSS Workshop on Trustworthy Elections (WOTE)*, 2007.
15. I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security: 18th International Conference, FC 2014*, pages 436–454, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
16. J.-P. Flori, J. Plt, J.-R. Reinhard, and M. Eker. Diversity and transparency for ecc. Cryptology ePrint Archive, Report 2015/659, 2015. `http://eprint.iacr.org/`.
17. S. Haber and W. S. Stornetta. How to time-stamp a digital document. *J. of Cryptology*, 3(2):99–111, 1991.
18. E. Heilman. One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner (poster abstract). In *Financial Cryptography and Data Security: FC 2014 Workshops, BITCOIN and WAHC 2014*, pages 161–162, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
19. A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015. `http://eprint.iacr.org/2015/366`.
20. A. K. Lenstra, B. Wesolowski, P. Egger, and D. Tristan. Trx: a trusworthy random elliptic curve service. `http://trx.epf.ch`, 2015.
21. M. Lochter and J. Merkle. Elliptic curve cryptography (ECC) brainpool standard curves and curve generation. RFC 5639, 2010.
22. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system,`http://bitcoin.org/bitcoin.pdf`, 2009.
23. National Institute for Standards and Technology. FIPS PUB 186-2: Digital signature standard, 2000. `http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf`.
24. NIST randomness beacon. `https://beacon.nist.gov`, 2011.
25. C. Pierrot and B. Wesolowski. Malleability of the blockchain's entropy. Preprint available at `https://almasty.lip6.fr/~pierrot/papers/Malleability.pdf`, 2016.
26. M. O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256 – 267, 1983.
27. RANDOM.ORG. `https://www.random.org`, 1998.
28. V. Rijmen and B. Preneel. A family of trapdoor ciphers. In E. Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 139–148. Springer Berlin Heidelberg, 1997.
29. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.
30. D. Simmons. US lottery security boss charged with fixing draw. `http://www.bbc.com/news/technology-32301117`, 2015.
31. Wikipedia. Pennsylvania lottery scandal. `http://en.wikipedia.org/wiki/1980_Pennsylvania_Lottery_scandal`, 1980.
32. Wikipedia. The truman show delusion. `https://en.wikipedia.org/wiki/The_Truman_Show_delusion`, 2016.
33. G. Wood. Ethereum: A secure decentralized transaction ledger. `http://gavwood.com/paper.pdf`, 2014.