A public randomness service

1 Introduction

The theoretical community has developed many clever cryptographic security protocols over the years for access, authentication, privacy, and authorization in networking and e-commerce applications. However, except for the simplest and most basic protocols, few have been widely deployed. A major reason concerns efficiency. Many of the more sophisticated security protocols, such as Zero Knowledge proof systems, are highly interactive and require too many communication rounds to be feasible in most situations. Other privacy-preserving protocols eliminate the need for many rounds of communication but assume the availability of a trusted source of randomness, an assumption that is not generally valid at present. We argue that it is time to design, implement, and deploy a trusted public randomness server on the Internet. The "NIST Beacon" [1] project aims at doing just that, starting with a prototype in 2011,

Trust is a complex concept, involving technical as well as social components. At the technical level, three fundamental elements of trust will be provided: *unpredictability*, *autonomy*, and *consistency*. Unpredictability means that users cannot algorithmically predict bits before they are made available by the source. Autonomy means that the source is resistant to attempts by outside parties to alter the distribution of the random bits. Consistency means that a set of users can access the source in such a way that they are confident that they all receive the same random string.

Experience teaches us that it is extremely hard to secure any single point in the Internet. However, it would be hard to simultaneously compromise the integrity of several servers. Therefore, we see our role as proposing a format that can be emulated by others so that the end result is several independent servers available to users. The participants on any multi-party computation protocol could then use the XOR of all sources. In that way, no one source would have to be trusted by all.

2 A Trusted Public Randomness Service

The most basic service a trusted public randomness service provides is to post a (pseudo-) random bit string S_L of a fixed length L, digitally signed by the server. The difference between a random string obtained from the randomness service and one

[1] Rabin appears to have first proposed this type of service, calling it a "beacon" [12]

generated locally by flipping coins is that the former is widely trusted to have resulted from a prescribed random process and was not "cooked" or otherwise falsified. A trusted string is known, by axiom, to have been unpredictable before it was produced.

2.1 A Simple Authorization Mechanism

The following situation occurs frequently in modern cryptographic applications:

- User P claims to have authorization to pass through a security checkpoint.
- The sentinel V says to P, "Authorized users are those able to invert the function f. Here is a challenge number y. Please tell me what $f^{-1}(y)$ is."
- P calculates and sends back $x = f^{-1}(y)$ to V. Then V checks that $f(x) = y$ and, if so, allows P to pass.

The terms "sentinel", "security checkpoint", and "user" are abstractions. The user can be a program, a person holding a smartcard, a client accessing a server, etc. The security checkpoint can be a physical place, an operating system security mechanism, an ATM, etc. The sentinel can be an operating system or one of its subsystems, an algorithm running on a network component, a human guard, etc.

This protocol uses the notion of a trap-door one-way function. Such a function f has the property that, given an arbitrary input x, $f(x)$ can be easily computed. However, only a user who knows the secret trap-door information can invert f in reasonable time.

Desired Properties User P is assumed not to be trusted. The purpose of this protocol is to prevent a corrupted P from getting past the sentinel without actually knowing the secret trap-door information. In many potential applications, the sentinel V also cannot be trusted. A corrupted V might try to obtain the secret trap-door information in order to enable him to bypass security mechanisms not only at his station, but at any other station using the same one-way function. Even if the sentinel can be trusted, if the sentinel knows the secret trap-door information, then it is necessary to protect the sentinel's data from the outside world. Depending on the application, this may be hard, expensive, or even impossible to do. Thus, one would like to implement this protocol in such a way that the sentinel does not need to know the trap-door information. Furthermore, we would like the trap-door to remain secret after the execution of the protocol so that the mechanism can be safely reused.

The beautiful insight, that perhaps it is not necessary for the sentinel to know the trap-door, is attributed to Von Neumann.[2] Clearly, only P needs to know the trap-door secret in order to carry out this protocol, for only P inverts f. The sentinel V only needs the ability to generate suitable challenge numbers and to compute f in order to verify the challenge response. But before we can be assured of the safety

[2] He, of course, did not pose the problem in these terms, as the notion of trap-door one-way function is quite recent.

of this protocol, we must ensure that V doesn't inadvertently learn the trap-door secret during the protocol's execution. Moreover, this should remain true even if V is corrupt and is maliciously trying to compromise P's secret.

Use of Randomness A question that comes to mind when considering the security of this protocol is, "How does the sentinel V choose the number y?" Clearly, if y is a fixed value, then anybody who witnessed an execution of the protocol could later pass through the checkpoint, as he would know $f^{-1}(y)$. Therefore, the sentinel must change y in each iteration of the protocol. But a simple change of y (such as adding 1 to the previously-used challenge) might allow special properties of f to be used to compute $f^{-1}(y)$ from previously known values of $f^{-1}(y)$. Ideally, from the point of view of preventing a corrupt P from passing the checkpoint, the number y should be chosen uniformly at random from a large set of "hard to invert" numbers. Thus, we must provide the sentinel with a good random (or pseudo-random) number generator. Fortunately, this is not hard to do. But neither is it trivial. For example, linear congruential generators turn out to be predictable by polynomial-time algorithms [3].

Another, less obvious, question that arises is, "How does the user P know that the sentinel chooses y at random, and why does he care?" In general, the user has no way of knowing, but it is important that the protocol be designed in such a way that the secret trap-door information remains secret regardless of how a corrupt P chooses y.

An Insecure Implementation Consider the following implementation of a trap-door function

- N is a product of two large primes p and q, each congruent to 3 modulo 4. Such a number is called a "Blum Integer". The trap-door is the prime p. It is known by authorized users but not by sentinels. N is publicly known.
- $f(x) = x^2 \bmod N$.

To pass the checkpoint, users must be able to demonstrate the ability to compute square roots modulo N. Since not all numbers have modular square roots, we must first solve the problem of how a sentinel should produce a random challenge. For Blum integers it is easy to find a number α modulo N such that for all $y \in Z_N^*$, exactly one of the four numbers in the set $C_y = \{\pm y \bmod N, \pm \alpha y \bmod N\}$ has a modular square root.[3] [4] The challenge issued by the sentinel to the party who wants to prove its knowledge of the factors of N is simply a number y chosen uniformly at random from Z_N^*. The response should be a modular square root x of the element in C_y that is guaranteed to have a modular square root. To verify that x is a valid

[3] Z_N^* is the group of invertible elements modulo N.

[4] The reader acquainted with Number Theory will recognize that α can be any number with Jacobi symbol -1 modulo N. Such a number is easily found in probabilistic polynomial time without having to factor N.

response to the challenge, the sentinel only needs to compute $x^2 \bmod N$ and test for membership in the set C_y.

It can be formally shown that users reveal no information about the trap-door by responding to a challenge generated according to this protocol. If the sentinel does not know the factorization of N then it will not be able to compute the necessary square root. A simple lemma, due to Rabin, shows that the problem of computing modular square roots is computationally equivalent to the problem of integer factorization. Thus, the usual assumption about the intractability of factoring implies that computing modular square roots is intractable without the factors.

We have just described an identification mechanism that works provided all parties follow the protocol. The problem, in many applications, is that neither party can assume the other is acting honestly. For example, the sentinel may be after the trap-door secret and may not follow the protocol in generating y. Such a sentinel can discover the factorization of N with high probability, rendering this protocol absolutely insecure. Here's how. The dishonest sentinel generates y by choosing u at random from Z_N^* and then choosing y at random from the set C_{u^2}. It is easily shown that the numbers y chosen in this way are uniformly distributed over Z_N^*; hence, it is undetectable by P that V is not following the protocol. The value x that P returns satisfies $x^2 = u^2 \bmod N$. With probability 0.5, $x \neq u \bmod N$, in which case $\gcd(x+u, N)$ is a proper factor of N. The gcd is an easy computation; therefore the trap-door will not remain secret for long from this cheating sentinel.

To summarize, the failure of this protocol came about because P could not detect that V picked the number y in a special way that gave her additional information about y. However, the protocol does work correctly if both parties can trust that the challenge y is an exogenously generated random number. This is precisely what a public trusted randomness server provides. If we simply change the protocol so that in the second step, P and V obtain y from the trusted randomness server, then the protocol indeed becomes secure.[5]

The above example, aside from whatever merits it may have as an identification scheme, was constructed to introduce the main practical issue addressed by this paper:

> *Access to a common trusted source of randomness can make simple protocols secure that otherwise would not be.*

3 Applications of a Trusted Public Randomness Server

Providing network security and reliability requires the use of *cryptographic primitives.* Examples of such primitives are encryption, decryption, and digital signatures.

[5] In practice, we would probably use an off-line version of this protocol. The services provided by our server will also allow for this. Time-stamps, signatures and possibly other features can be used in order to guard against replay attacks.

Over the last three decades, cryptographers have identified a number of other primitives as being powerful tools for developing secure network applications. Examples of these are bit-commitment (see [4, 5, 6, 7, 10]), oblivious transfer (see [8, 11]), digital coin-flipping [2], cryptographically secure pseudo-random number generators [1], and zero-knowledge (ZK) proofs [7, 9]. The latter primitive implies the ability to prove to a third party that a boolean function $f(x)$ is satisfiable without revealing a satisfying assignment. Furthermore, some instantiations of this primitive allow proving *knowledge* of a satisfying assignment x_0 for f without revealing x_0.

ZK proofs are interactive: the prover engages the other party (the "verifier") in a conversation. After the conversation is over, the verifier is convinced that $f(x)$ is satisfiable but has not obtained any information besides this fact. Unfortunately, ZK proofs are usually impractical: they require too much interaction and involve too much communication and computation. This accounts for the fact that there exist very few applications of this tool currently in use.

There are a number of variants of ZK proofs in which the interaction is minimized both in total number of bits communicated and in number of rounds. Among these, the most practical protocols assume, in one way or another, access to a common random string.

There are a number of design and implementation issues that need to be addressed. Some of them are the following

- source of entropy;
- rate: how many bits per second;
- user interface;
- full-entropy strings or cryptographically secure pseudo-random strings;
- authentication method;
- time-stamping method;
- archival properties (e.g. can old strings be authenticated?);
- trust model: what, exactly, can the consumer assume?

At this moment we are thinking of broadcasting full-entropy bit-strings. We plan to post them in blocks of 256 bits per second. We intend to sign and time-stamp the bit-strings. We also plan to link the sequence of blocks with a secure hash so that it will not be possible, even for the source itself, to retroactively change a block without detection. As for source of entropy, we are talking to NIST physicists. Although the most important property of the randomness server is its independence from users (i.e. almost-perfectly random strings are good enough, provided they are unpredictable), we see no reason not to use the most sophisticated entropy source we can afford.

References

[1] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[2] Manuel Blum. Coin flipping by telephone. In *IEEE COMPCON*, pages 133–137, 1982.

[3] J. Boyar. Inferring sequences produced by pseudo-random number generators. *Journal of the Association for Computing Machinery*, 1989. To appear.

[4] J. Boyar, M. Krentel, and S. Kurtz. A discrete logarithm implementation of zero-knowledge blobs. *Journal of Cryptology*, 2(2):63–76, 1990.

[5] J. Boyar, C. Lund, and R. Peralta. On the communication complexity of zero-knowledge proofs. *Journal of Cryptology*, 6(2):65–85, 1993.

[6] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37:156–189, 1988.

[7] G. Brassard and C. Crépeau. Zero-knowledge simulation of boolean circuits. In *Advances in Cryptology - Proceedings of CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 223–233. Springer-Verlag, 1987.

[8] M. Fischer, S. Micali, C. Rackoff, and K.D. Wittenberg. An oblivious transfer protocol equivalent to factoring. presented at EuroCrypt 84 and at the NSF Workshop on the Mathematical Theory of Security, Endicott House, 1985.

[9] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all alnguages in NP have zero-knowledge proof systems. *JACM*, 38:691–729, 1991.

[10] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[11] J. Halpern and M. Rabin. A logic to reason about likelihood. In *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing*, pages 310–319, 1983.

[12] Michael O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983.