A Simple Method for Generating and Sharing Pseudo-Random Functions, with Applications to Clipper-like Key Escrow Systems

Silvio Micali[1] and Ray Sidney[2] *

[1] MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139.
[2] Trusted Information Systems, Inc., 3060 Washington Rd. (Rt. 97), Glenwood, MD 21738. e-mail: sidney@tis.com.

Abstract. We present a very simple method for generating a shared pseudo-random function from a poly-random collection of functions. We discuss the applications of our construction to key escrow.

1 Introduction

Much work has been done towards the general goal of enabling parties to share information and cryptographic capabilities. In particular, see Blakley's [4] and Shamir's [30] notion of *secret sharing*; Chor, Goldwasser, Micali, and Awerbuch's [8] notion of *verifiable secret sharing*; Desmedt and Frankel's [12] notion of *shared authenticators and signatures*; and De Santis, Desmedt, Frankel, and Yung's [9] notion of a *shareable function*. (A more extensive treatment of sharing cryptographic capabilities can be found in Desmedt's survey [13].)

In this paper, we describe a very simple method for generating and sharing a pseudo-random function $f(\cdot)$ among n players. That is, we generate a function $f(\cdot)$ such that:

- For all input x, any sufficiently large collection of players (at least u players) can compute and reveal $f(x)$.
- For all inputs x such that $f(x)$ has never been revealed, no sufficiently small collection of players (t or fewer players) is feasibly able to compute $f(x)$.

If the number of players is not large, our method is particularly attractive. Indeed, after a preprocessing step has been performed, each evaluation of f at an input x is obtained by essentially the following procedure:

1. Each player performs *locally* (i.e., noninteractively) some pseudo-random function evaluations.
2. These local results are revealed (in a single round of communication).
3. The party evaluating the function takes majority and exclusive-ORs of the revealed results.

Our method is readily applicable to key-escrow systems.

* This research was performed while this author was at MIT, supported by a Fannie and John Hertz Foundation Fellowship for Graduate Study.

D. Coppersmith (Ed.): Advances in Cryptology - CRYPTO '95, LNCS 963, pp. 185-196, 1995.

2 Background: Poly-Random Functions

We discuss briefly Goldreich, Goldwasser, and Micali's notion of a *poly-random collection of functions* [18]. For any $j \in \mathbb{N}$, set $I_j = \{0,1\}^j$. Informally, a *poly-random collection of functions* is a collection $F = \{F_j | j \in \mathbb{N}\}$, where $F_j = \{f_x | x \in I_j\}$ is a collection of 2^j functions, each mapping I_j to itself, satisfying the following properties:

- There is a polynomial-time algorithm A which, given inputs x and y satisfying $|x| = |y|$, computes $f_x(y)$.
- If we choose $x \in I_j$ at random, then the function $f_x : I_j \to I_j$ behaves (as far as a tester limited to computational time which is polynomial in j can tell) exactly like a "random" function mapping I_j to itself.

The precise definition is in [18], as is an efficient construction of a poly-random collection of functions from any cryptographically strong bit generator (CSB generator). Since Levin [27] and Impagliazzo, Levin, and Luby [22] showed how to construct CSB generators from one-way functions, we can assume that we have a poly-random collection of functions at our disposal (to have secure encryption, we require a one-way function).

In practice, something simpler and even easier to compute is often used as a "random function" in implementing protocols. If we want a collection of random functions mapping $I_a \to I_b$, we take a secure hash function H with range I_b, and set $f_x(y) = H(x \circ y)$, where $\circ$ denotes concatenation. If H is indeed a good hash function, and x is chosen at random, then we would expect $f_x(y)$ to behave like a random function.

3 Generating a Shared Pseudo-Random Function

Parts of our algorithms bear a resemblance to protocols developed in [32], [6], [1], and [16], which actually aimed at a somewhat different goal: namely, achieving "common coin flipping" protocols in Byzantine agreement scenarios.

3.1 Resilient Collections

Our method is based on a combinatorial object which we call a *resilient collection*. A resilient collection is what one gets by taking a classic combinatorial *covering design* and complementing each set in it.

DEFINITION. Let $0 \leq t < u \leq n$. An (n,t,u)-*resilient collection* of sets is a collection $\mathcal{S} = \{S_1, \ldots, S_d\}$ of subsets of $\{1, \ldots, n\}$ such that:

- $|S_i| = n - u + 1$ for $i = 1, \ldots, n$.
- If $S \subseteq \{1, \ldots, n\}$ and $|S| = t$, then there is an S_i such that $S_i \cap S = \emptyset$.

In section 3.3, we discuss how to produce resilient collections.

3.2 Constructing Shared Pseudo-Random Functions from Resilient Collections

We now outline our method. We shall fill in the details of our protocols in section 3.4.

Let F be a poly-random collection of functions, as discussed in section 2. Let $\mathcal{S} = \{S_1, \ldots, S_d\}$ be an (n, t, u)-resilient collection, and let $j \in \mathbb{N}$. We wish to produce a pseudo-random function mapping I_j to I_j, shared among n players. We make the natural identification between players and elements of $\{1, \ldots, n\}$.

Seed Generation and Distribution. Each subset $S_i \in \mathcal{S}$ of players jointly runs a protocol GEN-SEED(i) which chooses at random a j-bit secret, s_i, and makes it known to every player in S_i (but to no other players). We call s_i a "random function seed," or just "seed" for short.

Function Evaluation. The shared random function is just

$$f(y) = \bigoplus_i f_{s_i}(y).$$

That is, $f(\cdot)$ is the exclusive-OR of a group of pseudo-random functions (one function for each random function seed). The values $f_{s_1}(y), f_{s_2}(y), \ldots, f_{s_d}(y)$ are called the *pieces* of $f(y)$.

Note that, as desired, the definition of a resilient collection ensures that any u players have enough information to compute all values of $f(\cdot)$, but no t players have any information about the values of $f(\cdot)$. Even if the information that t players have is combined with knowledge of previously evaluated values of $f(\cdot)$, the properties of poly-random functions guarantee that the players have no *useful* information about new values of $f(\cdot)$.

What's required to complete our description are a method to select the collection $\mathcal{S}$, and more precisely specified protocols.

3.3 Existence and Construction of Resilient Collections

We now discuss choosing the collection $\mathcal{S}$ used in our construction. Before we give any constructions for $\mathcal{S}$, we first recall a lower bound on $d = |\mathcal{S}|$ due to Schonheim. Define $(a)_b$ to be the falling factorial:

$$(a)_b = (a)(a-1)\cdots(a-b+1) = \frac{a!}{(a-b)!}.$$

Theorem 1. *If d is the number of subsets in an (n, t, u)-resilient collection, then*

$$d \geq \frac{(n)_t}{(u-1)_t}.$$

Proof. Let $\mathcal{S} = \{S_1, \ldots, S_d\}$ be an (n, t, u)-resilient collection. Consider a particular subset S_i. There are

$$\binom{n - |S_i|}{t} = \binom{n - (n - u + 1)}{t} = \binom{u - 1}{t}$$

subsets of $\{1, \ldots, n\}$ of size t which are disjoint from S_i. Since there are $\binom{n}{t}$ subsets of $\{1, \ldots, n\}$ of size t, each of which must be disjoint from some S_i, the number d of subsets in $\mathcal{S}$ must be at least

$$\frac{\binom{n}{t}}{\binom{u-1}{t}} = \frac{(n)_t}{(u-1)_t}.$$

In the other direction, we now wish to derive an upper bound on how large d must be. Bounds of this type exist in the combinatorics literature (see, for instance, [31], for some asymptotic results), but for our purposes, we want the following specific result:

Theorem 2. *Set*

$$d = \left\lceil \frac{(n)_t}{(u-1)_t} \ln \binom{n}{t} \right\rceil.$$

Then there exists an (n, t, u)-resilient collection $\mathcal{S}$ of d sets, $\mathcal{S} = \{S_1, \ldots, S_d\}$.

Proof. We perform a probabilistic construction. Choose the d subsets $S_1, \ldots, S_d$ independently and uniformly at random to be subsets of size $(n - u + 1)$ of $\{1, \ldots, n\}$. Now, let S be any fixed t-element subset of $\{1, \ldots, n\}$. Consider a particular subset, S_i.

$$\Pr(S \text{ and } S_i \text{ are disjoint}) = \frac{\binom{n-t}{n-u+1}}{\binom{n}{n-u+1}}.$$

Since there are d subsets S_i, all distributed independently,

$$\Pr(S \text{ is not disjoint from any } S_i) = \left[1 - \frac{\binom{n-t}{n-u+1}}{\binom{n}{n-u+1}}\right]^d.$$

Because there are $\binom{n}{t}$ possibilities for S,

$$\Pr(\text{There is an } S \text{ which intersects every } S_i) \leq \binom{n}{t}\left[1 - \frac{\binom{n-t}{n-u+1}}{\binom{n}{n-u+1}}\right]^d.$$

Taking natural logs of both sides of this inequality,

$$\ln \Pr(\text{There is an } S \text{ which intersects every } S_i) \leq \ln \binom{n}{t} + d \cdot \ln\left[1 - \frac{\binom{n-t}{n-u+1}}{\binom{n}{n-u+1}}\right]$$

$$
\begin{aligned}
&< \ln \binom{n}{t} + d \cdot \left[-\frac{\binom{n-t}{n-u+1}}{\binom{n}{n-u+1}} \right] \\
&\leq \ln \binom{n}{t} - \left[\frac{(n)_t}{(u-1)_t} \ln \binom{n}{t} \right] \cdot \frac{\binom{n-t}{n-u+1}}{\binom{n}{n-u+1}} \\
&= 0,
\end{aligned}
$$

where the strict inequality arises from the fact that $e^x \geq 1 + x$, with equality if and only if $x = 0$.

Hence, with nonzero probability, no subset of $\{1, \ldots, n\}$ of size t intersects every S_i. So there must exist some particular $\mathcal{S}$ such that this is true; this $\mathcal{S}$ is a (n, t, u)-resilient collection.

It is worth mentioning that since $\binom{n}{t} < 2^n$, the lower bound and upper bound which we have just shown differ from each other by at most a factor of $(n \ln 2)$, and so our randomized proof of an upper bound is actually not too far from optimal. Now, it would be nice if, given legal values for n, t, and u, we could easily come up with an (n, t, u)-resilient collection of as few sets as possible. Unfortunately, we don't know how to do this, and it seems to be a difficult problem to solve in general. We give here two specific construction methods for producing families of resilient collections; in each construction, values for n and u can be specified, and an $(n, f(n, u), u)$-resilient collection is produced.

Method 1. Fix n and $1 \leq u \leq n$. Set

$$S_1 = \{1, \ldots, (n-u+1)\},\ S_2 = \{(n-u+1)+1, \ldots, 2(n-u+1)\},\ \ldots.$$

Here, each subset contains the next $(n-u+1)$ numbers, and we take enough subsets that every number between 1 and n is in at least one subset. Note that unless $(n-u+1)$ divides n, the last subset in this list will "wrap around" and include numbers which are already in S_1. It's not difficult to see that this method yields a collection of $d = \lceil \frac{n}{n-u+1} \rceil$ subsets which is either $(n, \lceil \frac{n}{n-u+1} \rceil - 2, u)$-resilient (if $(n-u+1)$ doesn't divide n) or $(n, \frac{n}{n-u+1} - 1, u)$-resilient (if $(n-u+1)$ does divide n).

Method 2. Again, fix n and $1 \leq u \leq n$. Set

$$S_1 = \{1, \ldots, (u-1)\}^{\mathrm{c}},\ S_2 = \{(u-1)+1, \ldots, 2(u-1)\}^{\mathrm{c}},\ \ldots,$$

where A^{c} denotes the complement (with respect to the set $\{1, \ldots, n\}$) of A. In analogy with method 1, we take enough subsets that each number between 1 and n is not contained in at least one subset, and we "wrap around" if $(u-1)$ doesn't divide n. This construction produces an $(n, 1, u)$-resilient collection consisting of $\lceil \frac{n}{u-1} \rceil$ subsets.

3.4 Protocols

We now give more specific protocols for generating and evaluating shared pseudo-random functions. We assume each player is capable of digitally signing its messages, and of communicating privately with every other player. We give protocols for dealing with a single specific random function seed, s_i; these protocols should be executed concurrently for each seed. We assume that all players behave properly during the seed generation and distribution phase[3].

Seed Generation and Distribution. We implement protocol GEN-SEED(i) (to generate a random seed s_i and distribute it to the set S_i of players) as follows. Fix some particular player $T \in S_i$ (for example, T could be the "first" player in S_i, in some ordering). T chooses s_i at random, and sends it to every player in S_i. Each player in S_i then takes the value s_i that it receives, signs the string $s_i \circ i$, and sends this signed message to every player in S_i.

Function Evaluation. To enable an entity E (E may or may not be one of the players who shared the pseudo-random function) to evaluate $f(\cdot)$ at a point y, each player in S_i simply sends the value $f_{s_i}(y)$ to E.

The definition of a resilient collection guarantees that E will receive each piece of $f(y)$. Nonetheless, this may not suffice to enable E to compute $f(y)$. If players can send spurious values for pieces of $f(y)$, then E might receive *multiple* candidates for a piece of $f(y)$. In general, in fact, E could receive up to $(n-u+1)$ distinct candidates for each piece, which would lead to up to $(n-u+1)^d$ potential values for $f(y)$. We give two solutions to this problem:

1. If $t < 2u - n$, then we can modify our construction by basing our protocols on an $(n, t, 2u-n)$-resilient collection, instead of on an (n, t, u)-resilient collection. If u players cooperate to reveal $f_{s_i}(y)$ to E, then this ensures that more than half of the players in S_i give E the *correct* value of $f_{s_i}(y)$. So for each piece of $f(y)$, E can just take the most popular value sent to it. This enables E to compute $f(y)$.
2. If we do not wish to use the above solution (either because $t \geq 2u - n$, or because we are unwilling to allow the increase in the number of subsets required to make an $(n, t, 2u-n)$-resilient collection), then we can have each player give E a *zero-knowledge proof* of the correctness of each alleged piece of $f(y)$ that it sends. This proof enables E to ascertain what the correct value for each key-piece is, and therefore, to reconstruct chip z's secret key.

The theoretical approach of the second solution requires some further elaboration. If a player $T \in S_i$ sends E the value c, how does he or she then give a zero-knowledge proof that $f_{s_i}(z) \circ i = c$? The answer lies in the fact that T has copies of $s_i \circ i$ signed by each player in S_i. Essentially, T needs to claim

[3] The second method in Appendix B can be used if this assumption is unwarranted.

possession of values s_i, V_1, V_2, ..., V_{n-u+1} such that $f_{s_i}(z) \circ i = c$ and V_a is a signature of $s_i \circ i$ relative to the public signing key of the player a in S_i, for all $1 \leq a \leq n - u + 1$. This claim is an NP statement, and hence, it is possible to perform a zero-knowledge proof of it, using the methods of Goldreich, Micali, and Wigderson [21], Brassard, Chaum, and Crépeau [7], or Feige, Fiat, and Shamir [14]. Any honest player can convincingly execute the zero-knowledge proof protocol, whereas a player who is trying to convince E of the correctness of some false value will only be able to do so with negligible probability (since it will not possess the digital signatures needed to perform the proof protocol correctly, and will be unable to forge them).

We observe that the complexity of the zero-knowledge based protocol can be reduced to one round by the adoption of *non-interactive zero-knowledge proofs* (See [10], [15], and [5] for more information), or by the idea of *hashing* to obtain challenges to the prover (see, for example, [17] or [2]).

Notice also that the zero-knowledge proofs need be used only if absolutely necessary. If all of the alleged values of $f_{s_i}(y)$ that E receives agree with one another, then E knows that it has the correct value for that piece of $f(y)$. It is only if E receives several distinct values for $f_{s_i}(y)$ that there is any need to engage in zero-knowledge proofs.

If the only way a player can fail to cooperate is to refuse to yield information (as opposed to being able to maliciously present false data), then we can, of course, forgo the zero-knowledge proofs. In addition, in this case, we do not need to implement the second round of communication in the seed generation and distribution phase, since there is no need for players to possess digitally signed seeds if any information they send is known a priori to be correct.

4 Shared Pseudo-Random Functions and Key Escrow

4.1 Background: Key Escrow

The past two decades of progress in hardware and software have made (presumably) secure cryptography feasible for a large segment of the population. Recently, this has generated much concern that governments and corporations will soon be unable to make use of certain tools that have traditionally been used to apprehend wrongdoers. In particular, it is feared that court-authorized wiretapping will become essentially useless as a means of law enforcement [11].

The obvious way to ensure that this doesn't happen is for governments or corporations to arrange to possess every user's secret key; however, even legitimate users then enjoy no privacy. A better alternative that has recently been proposed is *key escrow* (see [28], [29], [26], and [25] for examples). In a key escrow scheme, a user's secret key is somehow split into *shares* held by *trustees*. The intent is that these trustees may, under appropriate circumstances, enable the reconstruction of a given secret key; however, sufficiently few trustees, behaving maliciously, do not possess enough information to reconstruct any key.

The Clipper Chip. This scheme works in a "top-down" fashion. For every user, x, two trustees each generate, randomly and independently, a secret string. These two strings are then sent to x's chip, which exclusive-ORs them together to compute x's secret key, c. c is stored in tamper-proof memory, so that no one (not even x) has any information about what c is, except for the trustees. Whenever x wishes to send a message m to user y, using a common key K_{xy} (which x and y have previously agreed upon), x's chip sends not only the encryption of m with key K_{xy}, but also the encryption of K_{xy} with key c (the algorithm for this latter encryption is classified, and is referred to as *Skipjack*). When presented with authorization for a wiretap of x's communications, the trustees each reveal their share of c to the FBI, who is then able to decrypt first K_{xy}, and then m.

4.2 Bad Behavior of Trustees

We distinguish three types of bad trustees in a key escrow scheme:

1. *Gossipy* trustees, who try to procure information that they should not have about users' keys, or who spread users' key information about.
2. *Withholding* trustees, who do not cooperate with appropriate authorities to recover a user's secret key.
3. *Obstructive* trustees, who are so malicious that they may not even behave properly during the set-up phase of a key escrow scheme (when users generate or otherwise obtain their keys). We shall consider obstructive trustees to be both gossipy and withholding.

For $0 \leq t < u \leq n$, we define an (n, t, u)-*escrow scheme* to be a method of splitting a secret key among n trustees such that:

- If at most t trustees are gossipy, then reconstruction of a secret key is not feasible without a court order.
- If at most $(n - u)$ trustees are withholding, then reconstruction of a secret key can easily be accomplished with a court order.

With this terminology, the Escrowed Encryption Standard is a (2,1,2)-escrow scheme.

4.3 Our Escrow Scheme

Using our shared pseudo-random functions, we can easily create a Clipper-like (n, t, u)-escrow scheme, if no trustees behave obstructively[4]:

1. Each chip has a unique j-bit ID number.
2. The secret key of chip $\#z$ is the j-bit string $f(z)$, where $f(\cdot)$ is a pseudo-random function shared among the n trustees.

[4] We defer consideration of obstructive trustees until Appendix B.

3. To initialize a chip with its secret key, each trustee gives the chip all the pieces of $f(z)$ that it has (since no trustees are obstructive, no signatures or zero-knowledge proofs are necessary here).
4. To permit wiretapping of a chip's communications, the trustees perform a pseudo-random function evaluation protocol (with the FBI playing the part of the evaluator E). Two such protocols were presented earlier. However, if $(n - u + 1)^d$ is small (recall that this is an upper bound on the number of candidate values of the pseudo-random function), then an alternative protocol exists: have each trustee send all its key pieces to the FBI, and then let the FBI simply "cycle through" all possible secret keys, and see which one decrypts the user's messages.

5 Conclusion

We have presented a simple method of generating and sharing a pseudo-random function among a group of players. Our procedure can be used with any type of collection of pseudo-random functions: poly-random collections, hash functions, DES, etc. After an initial set-up phase, in which a shared function is generated, evaluating the function requires only one round of communication. Of course, communications grow in length as we increase the number of random function seeds; in some situations, exponentially many (in the number of trustees) seeds are needed.

Our shared pseudo-random functions can be readily applied to produce key escrow schemes. Since escrow schemes generally have a rather small number of trustees, the exponential number of seeds mentioned above is not a real problem for this application.

Like the Clipper Chip proposal, our escrow scheme also combines well with the key distribution scenarios suggested by Leighton and Micali in [26] to achieve a conventional cryptosystem which achieves many of the gains of public-key cryptography without requiring the complexity-theoretic assumptions needed for public-key cryptography.

References

1. B. Awerbuch, M. Blum, B. Chor, S. Goldwasser, and S. Micali. How to implement Bracha's $O(\lg n)$ Byzantine agreement algorithm. Submitted to 1985 Principles of Distributed Computing Conference.
2. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*. ACM, 1993.
3. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Advances in Cryptology— CRYPTO '88*, pages 27–35. Springer-Verlag, 1988.
4. G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the AFIPS 1979 National Computer Conference*, pages 313–317, June 1979. New York, NY.
5. M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *Siam Journal of Computing*, 20(6):1084–1118, December 1991.

6. G. Bracha. An $O(\lg n)$ expected rounds randomized Byzantine generals protocol. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 316–326. ACM, 1985.
7. G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.
8. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 383–395. IEEE, 1985.
9. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 522–533. ACM, 1994.
10. A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof-systems. In *Advances in Cryptology— CRYPTO '87*, pages 52–72. Springer-Verlag, 1987.
11. D. Denning. To tap or not to tap. *Communications of the ACM*, 36(3):25–44, March 1993.
12. Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In *Advances in Cryptology— CRYPTO '91*, pages 457–469. Springer-Verlag, 1991.
13. Y. G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications and Related Technologies*, 5(4):449–457, 1994.
14. U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.
15. U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero-knowledge proofs based on a single random string. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 308–317. IEEE, 1990.
16. P. Feldman and S. Micali. Byzantine agreement in constant expected time (and trustine no one). In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 267–276. IEEE, 1985.
17. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology— CRYPTO '86*, pages 186–194. Springer-Verlag, 1986.
18. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807, October 1986.
19. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 174–187. IEEE, 1986.
20. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM, 1987.
21. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the Association for Computing Machinery*, 38(1):691–729, July 1991.
22. R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 12–24. ACM, 1989.
23. J. Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 20–31. ACM, 1988.

24. J. Kilian. *Uses of Randomness in Algorithms and Protocols.* ACM Distinguished Dissertations. MIT Press, 1990.
25. J. Kilian and T. Leighton. Failsafe key escrow. Technical Report TR-636, MIT, August 1994.
26. T. Leighton and S. Micali. Secret-key agreement without public-key cryptography. In *Advances in Cryptology— CRYPTO '93*, pages 456–479. Springer-Verlag, 1993.
27. L. Levin. One-way functions and pseudorandom generators. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 363–365. ACM, 1985.
28. S. Micali. Fair cryptosystems. Technical Report TR-579.b, MIT, November 1993.
29. National Institute for Standards and Technology. *Escrowed Encryption Standard (EES)*, 1994. Federal Information Processing Standards Publication (FIPS PUB) 185.
30. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
31. J. Spencer. Asymptotically good coverings. *Pacific Journal of Mathematics*, 118(2):575–586, 1985.
32. A. Yao. On the succession problem for Byzantine generals. Manuscript, Stanford University.
33. A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.

Appendix A: Oblivious Trustees

In [28], Micali brings up an idea which he calls *oblivious trustees*— namely, that trustees who are cooperating with the FBI to reveal a user's secret key should not know *whose* secret key they are revealing. This prevents them from giving advance warning to the party whose line will be tapped. By using an *oblivious circuit evaluation* protocol[5], we can make our scheme have oblivious trustees. Note that oblivious circuit evaluation is a somewhat complicated process, and so this feature may be more theoretical than practical.

We assume that there is some court which is capable of signing authorizations for wiretaps. That is, if chip z is to be tapped, then the court signs z and gives this signature to the FBI. It is enough for us to consider how the FBI gets a hold of a single value $f_{s_i}(z)$ from a trustee $T \in S_i$.

Define

$$g(s_i, \text{SIGS}, z, \text{AUTH}) = \begin{cases} f_{s_i}(z) & \text{If SIGS consists of legal signatures of } s_i \\ & \text{by all trustees in } S_i \text{ and AUTH is a legal} \\ & \text{signature of } z \text{ by the court.} \\ \text{"ERROR"} & \text{Otherwise.} \end{cases}$$

[5] *Oblivious circuit evaluation* is a way for two parties, A and B, who hold private inputs x and y, respectively, to interact to compute a value $f(x, y)$. After they have finished interacting, B has learned $f(x, y)$, but nothing else about x, and A has learned nothing about y. See [33], [20], [23], and [24] for more information about oblivious circuit evaluation, including protocols for accomplishing it.

Then we perform an oblivious circuit evaluation in which the FBI computes g, where T supplies inputs s_i and SIGS, and the FBI supplies inputs z and AUTH. This gives the FBI precisely the information it deserves and needs to know.

Observe that to make our scheme have oblivious trustees, we needed to use the fact that users' secret keys are chosen using poly-random functions. If secret keys were not chosen in this fashion, there would be no underlying algebraic connection between them— that is, if users were given keys completely independently of each other, then there would be no small circuit which could compute chip z's secret key from the value z— and so trustees could not be made oblivious like this.

Appendix B: Obstructive Trustees

We now sketch methods for dealing with obstructive trustees. There are two possibilities to consider:

1. We merely want to be aware of the presence of obstructive trustees, so that we can abort setting up our cryptosystem (and possibly start over again, with different trustees). In this case, the protocols we presented work, as long as we insist that at each stage, each trustee and each chip checks the data it has and makes certain that it is consistent. For example, in the chip initialization phase, each chip should ensure that for each $1 \leq i \leq d$, it received a common value s_i from every trustee in S_i. If it didn't, then the chip should complain that there is a malicious trustee present.
2. We need to ensure that even if some trustees are obstructive, we can nevertheless go about setting up our cryptosystem. This is more complicated. In this case, we do something similar to what was done in the second solution from section 3.4. That is, we require that $t < 2u - n$, and base our key escrow scheme on an $(n, t, 2u - n)$-resilient collection. Since obstructive trustees are withholding (by definition), this means that in each S_i, the obstructive trustees are in the minority. Thus, the trustees in each S_i can use a general secure multi-party computation protocol (see [19], for example) to produce a common seed in the generate and distribute seeds phase. In addition, we modify the chip initialization phase so that for each seed i, the chip uses whichever alleged value has the most trustees claiming that it's correct; similarly, in the wiretapping phase, the FBI also takes the majority value for each individual key-piece. No digital signatures or zero-knowledge proofs are required.