A Survey of Pseudorandom Functions

Michela Meister
Stanford University

Abstract

We survey three pseudorandom function constructions – specifically the Goldreich-Goldwasser-Micali construction [3], the Naor-Reingold construction from pseudorandom synthesizers [4], and the canonical Naor-Reingold number-theoretic constructions from the DDH assumption and the assumption that factoring Blum integers is hard [5].

1 Introduction

Pseudorandom functions (PRFs) are a key cryptographic primitive, used in private-key exchange and authentication, among other areas. Informally, a PRF f is a deterministic algorithm that takes in a string x and for some random length-d string k, outputs $f(k,x)$, where $f(k,x)$ 'looks' random. However, what does it mean to 'look' random? Here we cannot simply consider individual functions – the definition of a PRF relies on distributions over functions, called function ensembles. Let F be a function ensemble with functions in the set $\{0,1\}^n \to \{0,1\}^n$, and let R be the function ensemble distributed uniformly over all functions in the set $\{0,1\}^n \to \{0,1\}^n$. Then F is a pseudorandom function ensemble, if there is no polynomial time algorithm, that, given the ability to query values on a function f', can determine with non-negligible advantage whether f' was drawn from F or from R. We define a pseudorandom number generator (PRG) in a similar way: A PRG $G : \{0,1\}^d \to \{0,1\}^n$ is a deterministic algorithm that takes in a seed $s \in \{0,1\}^d$ and outputs a string $G(s) \in \{0,1\}^n$ such that no polynomial-time algorithm can distinguish between a set of strings drawn at random from $\{0,1\}^n$ and a set of strings generated by drawing a set of random seeds from $\{0,1\}^d$ and applying the PRG G to each seed.

Present in all of these discussions is the need for efficiency. In addition to the pseudorandom requirements, PRFs need to be efficient to compute and evaluate in practice. This means that we want to minimize the description length of our functions, which we define as the number of bits needed to define the function outputs, while also choosing functions that can be computed efficiently.

2 Random Strings to Random Functions

The key insight that Goldreich, Goldwasser, and Micali demonstrate in [3] is that the pseudorandom string output from a PRG can be used to create pseudorandom functions. The construction works as follows: Let $G : \{0,1\}^d \to \{0,1\}^{2d}$ be a PRG and let $G(x) = G_0(x)||G_1(x)$, where we simply mean that $G_0(x)$ is the first half of the output and $G_1(x)$ is the second half. Now imagine x as the root of a tree structure, where $G_0(x)$ is the left-child of x, and $G_1(x)$ is the right-child. We can recurse on this structure by applying G to either of the children, that is, computing $G(G_0(x)) = G_0(G_0(x))||G_1(G_0(x))$. In this way, we can build a tree structure of whatever size we please. On a query q, we compute the function $f(q)$ by traversing the tree structure via the bits of q. We apply G to the value at our given node, and 'move' left or right, that is, we compute on the first or second half of the output $G(x)$ depending on the next bit of q: if the next bit is a 0, we compute on the first half, if the next bit is a 1, we compute on the second half. When we reach a leaf node, that is, when we have iterated through all the bits of q, we output the value computed at the leaf node. Observe that the time it takes to compute $f(q)$ is $|q| \cdot t_g$, where t_g is the time it takes to compute G on a string. Additionally, for every input q, because G is deterministic, $f(q)$ is consistent across multiply queries.

2.1 Security

How can we prove that the above construction is pseudorandom? The pseudorandomness of our PRF reduces

to the pseudorandomness of our PRG. The construction assumes that the PRG G outputs strings that are computationally indistinguishable from random. Thus, if there were some algorithm A that could distinguish the output of the PRF from random with non-negligible advantage, we could use A to break the pseudorandomness of G. Thus, assuming that PRGs exist, this construction is a secure PRF.

3 Pseudorandom Synthesizers

While the GGM construction relies on PRGs to construct PRFs, other cryptographic primitives can also be used to construct PRFs. In [4], Naor and Reingold introduce the pseudorandom synthesizer, which they use to give a parallel and more efficient construction for PRFs. At a high level, a pseudorandom synthesizer is a function that is pseudorandom even on a sequence of queries with some dependencies. We formally define a collection of pseudorandom synthesizers $S = \{S_2\}$ as follows: Let $X = \langle x_1, x_2, ..., x_k \rangle$ and $Y = \langle y_1, y_2, ...y_m \rangle$ be two sequences of polynomially-many uniformly-distributed n-bit strings. Each synthesizer $S_2 \in S$ operates on pairs, and for every pair (x_i, y_j), S_2 returns $S_2(x_i, y_j)$. We define $C_{S_2}(X,Y)$ to be the $k \times m$ matrix where cell (i, j) holds $S_2(x_i, y_j)$. If S is a collection of pseudorandom synthesizers, then for S_2 chosen randomly from S, there is no probabilistic polynomial time algorithm that can distinguish between the matrix $C_{S_2}(X,Y)$ and a $k \times m$ matrix of randomly-chosen l-bit strings with non-negligible advantage. Additionally, unlike PRGs, the output of a synthesizer is not required to be larger than its input.

As a note, it's important to see that not all PRGs are synthesizers. For example, let's consider a PRG G, where on input $s = x \circ y$, we have that $G(s) = G(x \circ y) = G'(x) \circ y$, where G' is some other PRG. The matrix $C_G(X,Y)$ would not look uniform at all – for example, a $\frac{1}{m}$-fraction of the strings in the matrix would have the suffix y_1. In this way, the requirements on synthesizers are stronger than they are for PRGs.

An extension of the synthesizer is the k-dimensional synthesizer, an efficient function on k inputs. A collection of k-dimensional synthesizers $S = \{S_k\}$ is pseudorandom if, given polynomially-many uniformly distributed values for each input, which we can represent as k sequences of length m $\langle X_1, X_2, ...X_k \rangle$, and for S_k drawn randomly from S, the matrix of outputs $S_k(x_1, x_2, ..., x_k)$ on all possible combinations of inputs $(x_1 \in X_1, x_2 \in X_2, ...x_k \in X_k)$ is computationally indistinguishable from random.

3.1 Parallel PRF Construction

So how do we build PRFs from synthesizers? The key idea is to use a recursive synthesizer, that, in each phase, halves the length of its input. Given a synthesizer $S_2 : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ we define the $2k$-dimensional synthesizer S_{2k} as follows, where S_{2k} takes a sequence of $2k$ n-bit strings as input: $S_{2k}(x_1, x_2, ...x_{2k-1}, x_{2k}) = S_k(S_2(x_1, x_2), S_2(x_3, x_4), ...S_2(x_{2k-1}, x_{2k}))$. Using this definition, for any n, we can compute $S_n(x_1, x_2, ..., x_n)$ using $\lceil \log n \rceil$ levels of recursion. In order to describe this operation for a specific synthesizer S_2 in the collection and a specific sequence of inputs $X = (x_1, x_2, ...x_{2k-1}, x_{2k})$, we define the squeeze of X with respect to S_2 as $SQ_{S_2}(X) = (S_2(x_1, x_2), S_2(x_3, x_4), ...S_2(x_{2k-1}, x_{2k}))$. The construction Naor and Reingold give uses a collection of pseudorandom n-dimensional synthesizers $S = \{S_n\}$, where, for some ordering of the synthesizers in the collection S, we call the j-th synthesizer S_n^j.

Given these definitions, we can now define the construction for an ensemble F of PRFs $f : \{0,1\}^n \to \{0,1\}^n$. The PRF requires a pair of keys $(\vec{a}, \vec{k})$, where $\vec{a} = (a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, ..., a_{n,0}, a_{n,1})$ is a sequence of $2n$ n-bit strings, and $\vec{k} = (k_1, k_2, ..., k_{\lceil \log n \rceil})$ is a list of $\log n$ indices into S, that is, using the elements of $\vec{k}$ to index into S, $\vec{k}$ defines a sequence of $\lceil \log n \rceil$ synthesizers $(S^{k_1}, S^{k_2}..., S^{k_{\lceil \log n \rceil}})$. This pair of keys then fully defines the PRF $f_{\vec{a},\vec{k}}$. On input x, $f_{\vec{a},\vec{k}}(x) = SQ_{S^{k_1}}(SQ_{S^{k_2}}(...SQ_{S^{k_{\lceil \log n \rceil}}}(\{a_{1,x_1}...a_{n,x_n}\})))$. In order to understand this definition, let's first examine the sequence $\{a_{1,x_1}...a_{n,x_n}\}$. Here the bits of x determine which n strings in $\vec{a}$ synthesizer $S^{k_{\lceil \log n \rceil}}$ will operate on in the first phase of recursion – if the i-th bit of x is a 0, then string $a_{i,0}$ is included, whereas if the i-th bit of x is a 1, string $a_{i,1}$ is included. Then in the first phase of the recursion, we compute $SQ_{S^{k_{\lceil \log n \rceil}}}(\{a_{1,x_1}...a_{n,x_n}\})$, squeezing a list of n strings to a list of $\frac{n}{2}$ strings. In each subsequent phase, the next synthesizer defined by $\vec{k}$ squeezes the output of its predecessor. Since we begin with n strings, and each phase halves the number of strings, we compute $f_{\vec{a},\vec{k}}(x)$ after $\lceil \log n \rceil$ phases. It is important to note that the construction as a whole is an n-dimensional synthesizer. This point is key – each input x selects a different combination of strings from $\vec{a}$, and so by the definition of an n-dimensional synthesizer, the output of the function on a polynomial number of inputs x is indistinguishable from random.

It may be helpful to visualize this process for computing $f_{\vec{a},\vec{k}}(x)$ as a binary tree, where the leaves are the list of strings $\vec{a}$, and there are edges from the leaves 'chosen' by x to a 'parent' string in the next level. That is, there

are edges from a_{1,x_1} and a_{2,x_2} to $S^{k_{\lceil \log n \rceil}}(a_{1,x_1}, a_{2,x_2})$. We continue to build the tree from the bottom all the way up to the root, by adding edges from every two strings x and y that S^{k_j} operates on to the output string $S^{k_j}(x,y)$. While this binary tree construction may seem very similar to the GGM construction, there are some important differences. First, in the GGM construction, the strings at each 'node' in the tree were completely determined by the PRG key, and the input x simply chose a specific leaf node in the tree. In contrast, in this construction, the strings at each 'node' are completely determined by the input x itself. More importantly, perhaps, the tree in the GGM construction has depth n, while this tree has depth $\log n$, making this construction far more efficient.

3.2 Efficiency

In order to discuss the efficiency of the construction in more detail, recall that the complexity class NC^i is the class of all problems that can be computed in $O(\log^i n)$ time on $poly(n)$ parallel machines. As described in the binary tree representation, the construction has only $\log n$ levels. Therefore, if the synthesizers for each phase are computable in NC^i, then each function f in the family of PRFs defined by this construction is computable in NC^{i+1}.

Additionally, the construction has a neat incremental quality: if we have already computed $f(x)$, for any y that differs from x in only 1 bit, we can compute $f(y)$ with only $\log n$ calls to pseudorandom synthesizers. We can see why this is true by examining the binary tree representation – flipping one bit of the input will only change the nodes that lie on the path from the leaf node corresponding to the flipped bit to the root, which includes only $\log n + 1$ nodes. Note that the GGM construction does not have such a property – a flipped bit may result in $\Theta(n)$ additional calls to a PRG.

3.3 Security

The proof idea for the security of the construction follows a hybrid argument. Let F be the ensemble of PRFs $f : \{0,1\}^n \to \{0,1\}^n$ defined by the construction. Let R be the ensemble of random functions $R_n : \{0,1\}^n \to \{0,1\}^n$. For each level j of recursion, we define H^j as the distribution of functions that are computed by starting j levels from the bottom of the tree and working towards the root. This means that H^0 represents the first level of recursion, that is, H^0 is the same distribution as F. Likewise, $H^{\lceil \log n \rceil}$ is the same distribution as R. Then it is possible to show that, if there is an efficient algorithm A that can distinguish between any two neighboring function distributions H^j and H^{j+1}, A can be used to build a distinguisher for a synthesizer S. From this, we conclude that, if there exists a probabilistic polynomial time algorithm B that can distinguish between F and R with non-negligible probability p, then there exists a probabilistic polynomial time algorithm C that is a distinguisher for a synthesizer S with non-negligible probability $\frac{p}{\lceil \log n \rceil}$. Thus, if pseudorandom synthesizers exist, then the PRF construction is secure.

4 Number-Theoretic Constructions

Following their work on PRFs from pseudorandom synthesizers, in [5] Naor and Reingold present two PRF constructions that are even more efficient and parallel, and still very simple to compute. The first construction discussed relies on the assumption that the Decision Diffie-Hellman (DDH) problem is hard, and the second construction requires that factoring Blum integers is hard.

4.1 PRFs from DDH

We first review the DDH assumption: Let P and Q be two primes, where Q divides $P-1$, and let Z_P^* be the multiplicative group modulo P of order Q. Let g be a generator for Z_P^*. Then the DDH assumption says that there is no probabilistic polynomial time algorithm that can distinguish between $\langle g, g^a, g^b, z \rangle$, where z is a random element from Z_P^*, and $\langle g, g^a, g^b, g^{ab} \rangle$ with non-negligible advantage. While the DDH assumption is only an assumption – there is no proof that such a distinguisher does not exist, there are arguments that support the assumption. For example, in [6] Shoup shows that there is no efficient algorithm that can solve the DDH problem with non-negligible advantage, given only black box access to the group. Here, black box access means that the algorithm is not allowed to see the representations of g, g^a, g^b, g^{ab}, but is only given 'pointers' to these values. So a probabilistic polynomial time algorithm that knows nothing about the group structure and must treat the group as a 'black box' cannot break the DDH problem.

4.1.1 Construction

We define the PRF construction as follows: each function f is defined by a key $\langle P, Q, g, \vec{a} \rangle$, where P and Q are primes as defined above, $\vec{a}$ is a sequence of $n+1$ elements of Z_Q, and g is an element of order Q in Z_P^*. On input x, where x is an n-bit string, we define $f_{P,Q,g,\vec{a}}(x) = (g^{a_0})^{\Pi_{x_i=1} a_i}$, that is, for each i, g^{a_0} is raised to the power a_i if $x_i = 1$. Note that, as described, the inputs to f are n-bit strings, and the range is $\langle g \rangle$, the set of elements generated by g. This is problematic, because all functions in a function ensemble ought to have the same domain and range, which is not the case here. However, via hashing

we can make the range of f uniform over $\{0,1\}^n$. To do this, we define a hash function h to hash the range of f to strings in $\{0,1\}^n$, and define a second function $f'_{P,Q,g,\vec{a},h} = h(f_{P,Q,g,\vec{a}}(x))$ as our PRF.

4.1.2 Efficiency

In order to compute $f_{P,Q,g,\vec{a}}(x))$, we only need two computations. First we need to compute the product $a' = a_0\Pi_{x_i=1}a_i$, where because g is of order Q, we can compute the product a' modulo Q. Then we need to compute $g^{a'}$. With some preprocessing, these computations are in TC^0, that is, $f_{P,Q,g,\vec{a}}(x))$ can be computed by circuits that have constant depth and polynomially-many gates.

4.1.3 Security

This PRF construction can be described using a version of the GGM construction, which will be important for undersanding the proof of security. Define the PRG $\tilde{G}_{P,Q,g,g^a}(g^b) = \langle \tilde{G^0}_{P,Q,g,g^a}(g^b), \tilde{G^1}_{P,Q,g,g^a}(g^b)\rangle = \langle g^b, g^{ab}\rangle$. Note that, given the key for G, that is, given a, we can compute $\tilde{G^1}$ efficiently. Then we can compute $f_{P,Q,g,g^a}(x)$ using $\tilde{G}$ and the GGM construction, with different values for g^a at each step. We define $f_{P,Q,g,\vec{a}}(x) = \tilde{G^{x_n}}_{P,Q,g,g^{a_n}}(...(\tilde{G^{x_2}}_{P,Q,g,g^{a_2}}(\tilde{G^{x_1}}_{P,Q,g,g^{a_1}}(g^{a_0}))))$. That is, in phase i, $\tilde{G^{x_i}}_{P,Q,g,g^{a_i}}(g^{b_i})$ outputs g^{b_i} if $x_i = 0$ and outputs $g^{a_i b_i}$ if $x_i = 1$. So after n phases, we have computed the same function $f_{P,Q,g,\vec{a}}(x)$ as we originally defined it.

We sketch the proof idea for the proof of security: Let F be the function ensemble defined by the PRF. Let R be uniformly distributed over all functions $R_n : \{0,1\}^n \to \{0,1\}^n$. It can be shown that a sequence of polynomially many samples $L_G = \langle \tilde{G}_{P,Q,g,g^a}(g_1^b)...\tilde{G}_{P,Q,g,g^a}(g_t^b)\rangle$ is pseudorandom iff any single sample is pseudorandom. Moreover, any algorithm that distinguishes between L_G and a random sequence can be made into a distinguisher for a single sample with the same advantage, that is, a distinguisher that breaks DDH with the same advantage. Then assume that there exists a probabilistic polynomial time algorithm A that distinguishes between F and R with advantage $> \frac{1}{n^c}$, for some $c > 0$, in time $t(n)$. Using A, it is possible to construct an algorithm D that distinguishes between the sequence L_G and a random sequence with advantage $> \frac{1}{n^{c+1}}$ in time $poly(n) \cdot t(n)$. Thus D can be made into a distinguisher that breaks DDH with advantage $\frac{1}{poly(n)}$. Therefore, the PRF is pseudorandom iff the DDH assumption holds.

4.2 PRFs from GDH

The Generalized Diffie Hellman (GDH) assumption says that, for P, Q, and g as defined above, given some sequence $\vec{a} = (a_1, a_2, ..., a_n)$ of elements in Z_Q and some subset $T \subseteq [n]$, there is no probabilistic polynomial time algorithm that, observing P, Q, and g, can compute $g^{\Pi_{i\in T} a_i}$ with non-negligible advantage after querying $g^{\Pi_{i\in T'} a_i}$ on only *strict* subsets $T' \subset T$.

4.2.1 Construction

In this construction the function $f : \{0,1\}^n \to \{0,1\}$ is defined by a key $\langle P, Q, g, \vec{a}, r\rangle$, where P, Q, and g are defined as above, $\vec{a} = (a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, ..., a_{n,0}, a_{n,1})$ is a sequence of elements in Z_Q and r is a random n-bit string. Then $f_{P,Q,g,\vec{a},r} = \langle g^{\Pi_{i=1}^n a_{i,x_i}}, r\rangle_2$, where $\langle g^{\Pi_{i=1}^n a_{i,x_i}}, r\rangle_2$ denotes the dot product of $g^{\Pi_{i=1}^n a_{i,x_i}}$ and r mod 2.

4.2.2 Security

We sketch the proof idea for the proof of security: Let F be the function ensemble defined by the PRF, and let R be uniformly distributed over all functions $R_n : \{0,1\}^n \to \{0,1\}$. Assume that a probabilistic polynomial time algorithm A exists that, observing P, Q, g, and r, can distinguish F from R with non-negligible advantage $> \frac{1}{n^c}$ in time $t(n)$. Then an algorithm D exists that uses A to distinguish $f_{P,Q,g,\vec{a},r}(1^n)$ from random with probability $> \frac{1}{n^c \cdot t(n)}$. Using the Goldreich-Levin hard-core bit theorem it can then be shown that this distinguisher D contradicts the GDH assumption [2], so as a result the PRF is pseudorandom if the GDH assumption holds.

4.2.3 n-Dimensional Synthesizers

As a note, one of the original motivations for constructing PRFs from the GDH assumption was to find a concise construction of an n-dimensional synthesizer – a construction without multiple levels of recursion. Using a similar construction as described above, we can define the synthesizer $S_{P,Q,g,r}(b_1, b_2, ...b_n) = \langle g^{\Pi_{i=1}^n b_i}, r\rangle_2$, where $(b_1, b_2, ...b_n)$ is a sequence of elements in Z_Q. Then, given k sequences $X_1, X_2, ...X_k$, the matrix of outputs $S(x_1, x_2, ...x_k)$ for any combination of inputs is computationally indistinguishable from uniform, and so $S_{P,Q,g,r}$ is an n-dimensional synthesizer.

4.2.4 PRFs from Factoring

A variant of the GDH assumption is the GDH assumption modulo a Blum Integer N. (A Blum integer N has the form $N = P \cdot Q$, where P and Q are primes and $P \equiv Q \equiv 3 \bmod 4$.) This assumption says that, for a Blum integer N, a quadratic residue g of Z_N^*, and some sequence $\vec{a} = (a_1, a_2, ...a_n)$ of elements in $[N]$ and some subset $T \subseteq [n]$, there is no efficient algorithm that, observing N and g can compute $g^{\Pi_{i\in T} a_i}$ with non-negligible advantage after querying $g^{\Pi_{i\in T'} a_i}$ on only *strict* subsets

$T' \subset T$. In [1], Biham, Boneh, and Reingold show that an algorithm that breaks GDH can be used to build an algorithm that factors Blum integers. Therefore, a PRF construction that is pseudorandom if the GDH assumption holds is pseudorandom if factoring Blum integers is hard. Additionally, this relationship is linear-preserving, which means that if an algorithm can break GDH in time $t(n)$ with advantage $\varepsilon(n)$, then there exists an algorithm that can factor Blum integers with the same advantage and running time $poly(n) \cdot t(n)$.

In this case the construction is nearly identical to the first construction from the GDH assumption, but where each function f is defined by a key $\langle N, g, \vec{a}, r \rangle$.

5 Conclusion

In this survey we explored the evolution of PRFs from the GGM construction based on PRGs, to synthesizer constructions, and finally to constructions that rely on the assumptions that DDH and factoring are hard.

Additionally, there are many other interesting applications of these PRF constructions to explore – for example, the number-theoretic constructions can be used for multi-party pseudorandom function evaluation, where the ability to query a function is distributed across multiple parties so that only certain sets of parties can evaluate the function. Other interesting applications include oblivious evaluations of pseudorandom functions.

Finally, another enticing topic for further research and surveying, perhaps outside of the cryptographic context, is the relationship between PRFs and computational learning theory. If a concept class contains pseudorandom functions, then there is no efficient algorithm that can learn that concept class. So PRFs can give us a better understanding of how hard it is to learn different concept classes.

References

[1] Eli Biham, Dan Boneh, and Omer Reingold. Breaking generalized diffie-hellmann modulo a composite is no easier than factoring. *Inf. Process. Lett.*, 70(2):83–87, 1999.

[2] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 25–32, New York, NY, USA, 1989. ACM.

[3] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[4] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. *Electronic Colloquium on Computational Complexity (ECCC)*, 2, 1995.

[5] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, March 2004.

[6] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.