Analysis of Underlying Assumptions in NIST DRBGs

Wilson Kan
Security and Privacy Group
Pitney Bowes Inc.*

September 4, 2007

Abstract

In [1], four different DRBGs are recommended for cryptographic purpose. Each generator is based on some underlying cryptographic concept. The article examines each of the concept to determine what are the necessary and sufficient conditions for the DRBG to be secured in its generation process. In addition, the effects of failure of typical cryptographic requirements of each underlying concept are discussed.

From [5], permutation based DRBGs are never indistinguishable from a true random source. From [4], elliptic based DRBGs are secured given a set of problems regarding elliptic curve remains difficult. This article demostrates that a pseudo-random family is required for both hash based and HMAC based DRBGs.

1 Introduction

Deterministic random bits generators (DRBG) are used in numerous applications ranging from gaming to cryptography. For various applications of DRBG, different properties of the DRBG are required. For example, DRBG in a video game requires very few properties, since there are no major consequences if bits are predictable. On the other hand, a DRBG used for online gambling would require that future bits to be unpredictable; otherwise, some players will have significant advantage over other players.
In this article, we examine DRBG used in cryptographic schemes. For example, using DRBG as a one-time pad or as a key generator. The requirement that one seeks in such a DRBG is that the DRBG is cryptographically strong.
This article examines various assumptions that one makes about the underlying cryptographic tools one uses for the DRBG recommended in [1] in relation to the security of the generation process of each DRBG.
In section 3, we see that collision resistance and pre-image resistance are not necessary for a cryptographically strong DRBG. However, the failure of either property can result in the failure of the necessary condition of pseudo-randomness.
In section 4, we see that resistance existential forgery is similar to collision resistance in the sense that it is not necessary but desired. Also, we observe that resistance to key recovery is necessary for the DRBG to be cryptographically strong. In addition, we note that the underlying hash function for the HMAC construct must be a pseudo-random family.
In section 5, we examine [4] and review the conclusion made by the author.
In section 6, we show the basic idea to why permutation based DRBG can never be cryptographically strong. This result is based on the analysis done in [5].

2 Security Definition

We consider a computationally bounded adversary, who knows everything about the DRBG except for the secret state(s). The goal of the adversary is to gain any information about the future bits or any unseen bits in the past.
We consider two security features for a DRBG.

*Consulting from Lisa Yin

Definition A DRBG is said to be *pseudo-random* if its outputs are indistinguishable from a truly random bit string of the same length. (See Appendix A for the definition of indistinguishable.)

Pseudo-randomness is the primary concern in this article. We will examine what are some of the necessary and sufficient conditions for a DRBG to be pseudo-random.

Definition A DRBG is said to have *forward secrecy* if a future state is known, the current bit string remains to be indistinguishable from a truly random bit string of the same length.

Forward secrecy of the DRBG in [1] is generally built into the generator by the property of the underlying function or problem.

Definition A DRBG is said to be *cryptographically strong* if it is pseudo-random and it has forward secrecy.

3 Hash DRBG

We will look at the consequences, with regards to the hash based DRBG in [1], of the failure of some typical assumptions that one makes about hash functions.

3.1 Notations

$\{0,1\}^k$ - a bit string of length k.

$\{0,1\}^*$ - a bit string of arbitrary length.

$hash(\cdot)$ - a hash function mapping from $\{0,1\}^*$ to $\{0,1\}^n$ for some fixed n.

n - the length of the output of $hash(\cdot)$.

3.2 Basic Construction

The simplified version of the hash based DRBG is as follow:

V is a secret state variable of the DRBG. And the random bits output is:

$$hash(V+1)||hash(V+2)||hash(V+3)||\dots$$

3.3 Underlying Assumptions

3.3.1 Collision Resistance

Definition $hash(\cdot)$ is said to be *collision resistance* if it is computationally infeasible to find $x, y \in \{0,1\}^*$ with $x \neq y$ such that $hash(x) = hash(y)$.

It is common to assume that a hash function is collision resistance when it is used for cryptographic purposes. However, this requirement does not seem to be necessary for the DRBG described in [1]. Firstly, a collision is unlikely to match the secret value. Secondly, even if one finds a collision for some valid secret value, it is not certain how the collision can be useful, since $hash(x) = hash(y)$ does not imply $hash(x+1) = hash(y+1)$ However, collision resistance is still a desired property in the underlying hash function, since it can interfere with the pseudo-random property of a hash function.

3.3.2 Pre-image Resistance

Definition $hash(\cdot)$ is said to be *pre-image resistance* if, given $y \in \{0,1\}^n$, it is computationally infeasible to find $x \in \{0,1\}^*$ such that $hash(x) = y$.

Suppose there exists an attacker against a DRBG using a hash function with 128 bits security that, given $y \in \{0,1\}^{256}$, the attacker can always find $x_1 \in \{0,1\}^{440}$ such that $hash(x_1) = y$. Moreover, assume that whenever the attacker has a set of distinct pre-images, $\{x_1, \ldots, x_n\}$, of y, then the attacker is able to find another pre-image $x_{n+1} \neq x_i$, for $1 \leq i \leq n$.
For simplicity, we will assume that the distribution of the outputs of the hash function is uniform. Then, given $y \in \{0,1\}^{256}$, the number of pre-images of y is $2^{440-256} = 2^{184}$. That is, the probability of the attacker getting the correct secret value is 2^{-184}. Thus, even if the attacker's method of finding pre-images is in constant time, it is still computationally infeasible to find the secret value. Similar argument can be carried out for using a hash function with 256 bits security.
As we can see, even given such a powerful attacker, the secret value is still infeasible to compute. Thus, pre-image resistance is not a necessary requirement of the hash function.
Similar to collision resistance, it is still desirable to have pre-image resistance as a property to prevent potential exploit, which might interfere with the pseudo-randomness.

3.3.3 Pseudo-random Family

Definition A *random oracle* is a black box that upon every input, an output is chosen uniformly from the codomain; except that for any specific input, it returns the same output.

Definition A family of functions is said to be a *pseudo-random family* if each function in the family together with a secret value is indistinguishable from a random oracle.

It is easy to show that if a hash function with its input as a secret value is a pseudo-random family, then the output of the hash DRBG recommended in [1] is indistinguishable from a true random source(see Appendix B.) Thus, pseudo-randomness is a sufficient condition for the security of the hash DRBG. However, proving pseudo-randomness of a hash function is difficult (and impossible in many cases). Two conditions for a function f such that the conditions together are equivalent to being indistinguishable from a random oracle are:

i - The output is uniform over the entire codomain. (*Uniformity*)
ii - $f(x_1, \ldots, x_n)$ is independent from $f(y_1 \ldots, y_n)$ if $x_i \neq y_i$ for some $i \in \{1, \ldots, n\}$. (*Independency*)

The equivalent holds from the definition of a pseudo-random family and random oracle.
Failure of either one of these conditions would clearly weaken or break the pseudo-randomness of the output.

3.4 Summary

Being a pseudo-randomness family is the only property that the hash function requires such that the final output of the DRBG is indistinguishable from a random bit string. Collision resistance and pre-image resistance are desirable in the sense that it can interfere being a pseudo-random family.
Forward secrecy is built into the DRBG since finding the state variable is the same as finding the correct pre-image of a hash value.
Care should be taken when this DRBG is implemented as passing statistical tests does not necessary mean that the hash function behaves as a pseudo-random family.

	Collision resistance	Pre-image resistance	Uniformity	Independency
Necessary conditions	✗	✗	✓	✓

4 HMAC DRBG

An HMAC has two inputs: a secret key and the message. Typically, when HMAC is used for message authentication, the message portion is assumed to be known to the attacker. However, in the HMAC DRBG described in [1], the "message" is a second secret value. The major difference between HMAC based DRBG and hash based DRBG is in the way that the random bits are produced.

4.1 Notations

n - the length of the output of the based hash function.

k - a secret key of length l.

$HMAC(k, \cdot)$ - an HMAC function mapping from $\{0,1\}^l \times \{0,1\}^*$ to $\{0,1\}^n$ as described in [9].

4.2 Basic Construction

The simplified version of the HMAC based DRBG is as follow:

K and V are secret state variables of the DRBG. Inductively define, $V_1 = V$ and $V_i = HMAC(K, V_{i-1})$. And the random bits output is:

$$HMAC(K, V_1)||HMAC(K, V_2)||HMAC(K, V_3)||\dots$$

4.3 Underlying Assumptions

4.3.1 Existential Forgery

Definition $HMAC(k, \cdot)$ is said to resist *existential forgery* if it is computationally infeasible to find $x, y \in \{0,1\}^*$ with $x \neq y$ such that $HMAC(k, x) = HMAC(k, y)$ for a fixed unknown k.

Similar to collision resistance, the ability to existentially forge a HMAC output does not seem to weaken the DRBG. A second "message" with the same HMAC value does not seem to be useful in finding additional information on the generated bits. However, as to hash based DRBG, it is a desirable property to prevent interference to the pseudo-random family property.

4.3.2 Key Recovery

Definition $HMAC(k, \cdot)$ is said to resist *key recovery* attacks if it is computationally infeasible to find k given a $HMAC(k, \cdot)$ oracle.

Suppose there exists an attacker that can efficiently compute k. If the attacker is given the first block $b_1 \in \{0,1\}^n$, by computing $b_i = HMAC(k, b_{i-1})$ for the i^{th} block, the attacker can completely predict the output bits generated by the HMAC DRBG.
This attack shows that resistance to key recovery is a necessary condition for the pseudo-randomness of the output.

4.3.3 Pseudo-randomness

Similar to the hash DRBG, pseudo-randomness plays a necessary role in the generation of pseudo-random outputs. It is shown in Appendix B that, given $HMAC(k_0, \cdot)$ is a pseudo-random family with k_0 as the secret value, then the output is pseudo-random.
It is shown in [3], that an HMAC is a pseudo-random family if the underlying hash function is a random oracle. On the other hand, if the underlying hash function is not a random oracle, it is shown in [6] that HMAC is susceptible to both forgery and key recovery attacks.

4.4 Summary

In addition to pseudo-randomness, resistance to key recovery attacks is a necessary property that the HMAC requires such that the final output of the DRBG is indistinguishable from a random bit string. Resistance to key recovery attacks also proves forward secrecy.
In order to obtain pseudo-randomness, a pseudo-random hash function is needed; in which case, one can use the hash DRBG to ensure pseudo-randomness.
The iterative method for generating the output does not provide additional security. In fact, the iterative process seems to give additional information to the adversary. Since each iteration uses the previous block as the "message", the message portion of each HMAC call is known entirely (with the exception for the first call.)

It does not seem like HMAC DRBG provides additional security nor efficiency to be used over a hash based DRBG.

	Resists existential forgery	Resists key recovery	Uniformity	Independency
Necessary conditions	✗	✓	✓	✓

5 Elliptic Curve DRBG

It is shown in [4] that if the x-Logarithm Problem, the Truncated Point Problem, and the Decisional Diffie-Hellman Problem are hard, and a slight alteration to the given truncation method recommended, then the elliptic curve DRBG recommended in [1] is pseudo-random under appropriate choices of elliptic curve.

5.1 Notations

$E(\mathbb{F}_q)$ - the elliptic curve group defined by the elliptic curve E over the Galois field $\mathbb{F}_q$.

P - an element of $E(\mathbb{F}_q)$ such that order of P is some large prime.

Q - a point in the subgroup generated by P.

n - order of P

$x(\cdot)$ - a function mapping from $E(\mathbb{F}_q)$ to $\{0,1\}^k$ by taking the x-coordinate and converting it to a binary string.

$t(\cdot)$ - a truncation function mapping from $\{0,1\}^k$ to $\{0,1\}^l$.

5.2 Basic Construction

The simplified version of the elliptic curve based DRBG is as follow:

s is a secret state variable of the DRBG. Inductively define, $s_0 = s$ and $s_i = x(s_{i-1}P)$. And the random bits output is:

$$t(x(s_1Q))||t(x(s_2Q))||t(x(s_3Q))||\dots$$

5.3 Underlying Assumptions

5.3.1 Elliptic Curve Discrete Logarithm Problem

Problem (ECDLP)**.** *Let z be a random integer uniformly distributed in the interval $[0, n-1]$. Given zP, determine the value of z.*

This is the underlying hard problem to all the problems considered in the security proof. ECDLP is widely believed to be difficult. However, if this problem is found to be easy, then there is no chance for any elliptic curve based cryptography to be secured. Note that ECDLP is easy for some choice of E, P and/or Q.

5.3.2 x-Logarithm Problem

Problem (XLP)**.** *Let d and z be random integers uniformly distributed in the interval $[0, n-1]$. Given R equals to dP or $x(zP)P$ both equally likely, determine with probability better than $\frac{1}{2}$ if $R = dP$ or $R = x(zP)P$.*

The hardness of this problem is a necessary condition in order to obtain forward secrecy. It is trivial to see that the XLP is polynomial-time reducible to ECDLP. However, whether ECDLP is polynomial-time reducible to XLP is an open question. If the open question is shown to be true, then we know XLP is hard. There is a stronger version of XLP described in [4], the AXLP, that can be shown under mild conditions that AXLP is at least as hard as DDHP, which suggests that XLP is also hard.

5.3.3 Truncated Point Problem

Problem (TPP). *Let t be an appropriate choice of truncation function. Let R be a random point in the subgroup generated by P. Let b be a random binary bit string with length equals to the output length of t. Given u equals to $t(x(R))$ or b both equally likely, determine with probability better than $\frac{1}{2}$ if $u = t(x(R))$ or $u = b$.*

The hardness of this problem is used to establish the pseudo-randomness of the output of the DRBG. One major concern with TPP is that $x(R)$ is an integer in $[0, n-1]$ instead of $[0, 2^k - 1]$ for some k. The problem is easiest to illustrate with the following example:

Example Let X be a random variable that generates uniform random variable in $[0, 100]$ and convert to a binary string of length 7.
Let B be a random variable that generates uniform random variable in $\{0, 1\}^7$.
Let t be a naive truncation function that keeps only the leftmost (most significant) bit.
It is easy to see that given a value x from X, $Prob[t(x) = 0] = 0.63 > \frac{1}{2}$.

In addition, since $x(R)$ is a point on $E(\mathbb{F}_q)$, there are additional structure that might be exploited. However, with these concerns in mind, when n is large and a better truncation function is chosen, the bias should be minimal. Note that the TPP associated with truncation function given in the NIST's recommendation (namely, removal of some of the leftmost bits) is easy. Also, some elliptic curve properties need to be considered for the appropriate choice of t. The choice of representation of $\mathbb{F}_q$ might give rise to fixed bits with some E and trace bits always exist for different $E(\mathbb{F}_q)$, which must be truncated.

5.3.4 Decisional Diffie-Hellman Problem

Problem (DDHP). *Let q, r and z be independent random integers uniformly distributed in $[0, n-1]$. Given (Q, R, S) equals to (qP, rP, qrP) or (qP, rP, zP) both equally likely, determine with probability better than $\frac{1}{2}$ if $(Q, R, S) = (qP, rP, qrP)$ or $(Q, R, S) = (qP, rP, zP)$.*

DDHP is the primary underlying hard problem for elliptic curve DRBG provided by [1]. DDHP along with TPP and XLP are sufficient to show that the DRBG is cryptographically strong. It is unknown whether DDHP is a necessary condition. DDHP is "easier" than the ECDLP. However, no efficient algorithm is known currently to solve this problem.

5.3.5 Computational Diffie-Hellman Problem

Problem (CDHP). *Let q and r are independent random integers uniformly distributed in $[0, n-1]$. Given the triplets (P, qP, rP), determine the value of qrP.*

It is shown in [8] that ECDLP and CDHP for various elliptic curves are polynomial-time reduction of one another. Thus, the same conclusion for ECDLP can be made for CDHP.
CDHP is necessary for the DRBG to be cryptographically strong. It is unknown if it is sufficient.

5.3.6 Other

If n is not a large prime, then elliptic curve DRBG is susceptible to small subgroup attack.
If E and/or P are poorly chosen, most of the above problems can be easy to solve.

5.4 Summary

ECDLP and DDHP have been studied for many decades and most scholars agree that CDHP and ECDLP are hard problems. XLP is a new problem that is closely related to ECDLP. There are evidents that XLP is true, but not confirmed as of now. As for TPP, in [4], there is a heuristic for estimating the statistical distance between the truncation and a random bit string. In short, truncating the appropriate bits, and truncating sufficient number of bits should provide sufficient randomness for a computationally bounded adversary.

	ECDLP	XLP	TPP	DDHP	CDHP
Necessary hard problems	✓	✓	✓	✗	✓

6 Block Cipher DRBG

This section is structured slightly differently due to the research done in [5].

6.1 Notations

k - a secret key of from the key space.

n - the length of every key in the key space.

$E(k, \cdot)$ - a symmetric encryption function mapping from $\{0,1\}^n \times \{0,1\}^L$ to $\{0,1\}^L$.

$D(k, \cdot)$ - a symmetric decryption function mapping from $\{0,1\}^n \times \{0,1\}^L$ to $\{0,1\}^L$ such that for all $x \in \{0,1\}^L$ and any $k \in \{0,1\}^n, D(k, E(k, x)) = x$.

L - the length of the output of the encryption function.

6.2 Basic Construction

The simplified version of the block cipher based DRBG is as follow:

K and V are secret state variables of the DRBG. And the random bits output is:

$$E(K, V+1)||E(K, V+2)||E(K, V+3)||\dots$$

6.3 Inherent Flaws

Definition $f : \{0,1\}^L \to \{0,1\}^L$ is said to be a *random permutation* if f is bijective and for each input, a fixed output is chosen uniformly from the codomain.

A major different between hash based DRBG and any encryption based DRBG is that encryption based DRBG uses a pseudo-random permutation (for any fixed key) instead of a pseudo-random family. Due to the properties of being a pseudo-random permutation, an inherent weakness exists and is illustrated with a simple example:

Example For simplicity, the example will attempt to generate a pseudo-random four digits number (we allow zero as a leading digit) using the same idea as the block cipher DRBG.
Let $E(k, \cdot) : \{0,1\}^n \times [0,9] \cap \mathbb{N} \to [0,9] \cap \mathbb{N}$ be a random permutation for any k.
For $k \in \{0,1\}^n$ and $r \in [0,9] \cap \mathbb{N}$, the corresponding "random" four digits number is $E(k, r+1)||E(k, r+2)||E(k, r+3)||E(k, r+4)$. Since $E(k, \cdot)$ is a random permutation, it is certain that the four digits must all be different.
This fact implies that only 5040 of all 10000 possible outcomes are possible outputs, which is far from random.

From the example, one can see that the size of the range space of $E(k, \cdot)$ and the length of the final output are critical factors for the randomness of the output. Essentially, the lack of randomness lies in the difference between a random oracle and a random permutation; specifically, a random permutation fails to satisfy independency. A random permutation must have different values for different inputs.
For further and more technical details about this flaw, read [2] and [5].

6.4 Underlying Assumptions

6.4.1 Pseudo-random Permutation

As described above, using a pseudo-random permutation in counter-mode, is not indistinguishable from a truly random source. One can propose a different method of generating the output without the use of counters. However, due to the nature of a pseudo-random permutation, it is impossible to avoid the situation in the example without resorting to some other random number generator.
One can see from the example (or from [5]) that the randomness of the output from the block cipher DRBG is proportional to the square of the output length and inversely proportional to L. Thus, requesting a short sequence of random bits from a block cipher that encrypts a long message minimizes the adversary's ability to distinguish the output from random.

6.4.2 Key Recovery

It is necessary that the block cipher resists key recovery attacks. If the adversary can efficiently compute the key k, then the adversary can simply run $D(k, \cdot)$ to find the counter used; thus, completely revealing the remaining bits.
The block cipher DRBG is partly designed specifically to resist key recovery attack, since the recommended block ciphers (TDEA and AES) are secured with the corresponding security levels.

6.5 Summary

The block cipher DRBG should be avoided generally. The various security levels of AES that are recommended for the use of the DRBG are only useful to prevent key recovery attacks. Since all the AESs have the same output length, the pseudo-randomness of the DRBG is independent of which AES is chosen.

7 Conclusion

In general, block cipher DRBG should not be used in any circumstances. If block cipher DRBG is currently implemented, then, if possible, one should change the DRBG immediately; otherwise, one should be certain that the outputs generated are as short as possible relative to the output block size of the block cipher used. The other three DRBGs are secured under the current knowledge we have about the underlying mechanisms. Elliptic curve DRBG is the most computationally expensive DRBG. However, it is also the one that is the best understood. ECDLP has been researched for many decades and is still believed to be a hard problem, whereas many hash functions are failing collision resistance as time passes. Two major advantages elliptic curve have over hash or HMAC based DRBG are that the maximum length of output is significantly greater and that it is a number theoretic based instead of heuristic based. If elliptic curve computations can be done as efficiently as hashing (via improve algorithms or hardware), then elliptic curve DRBG is currently the best DRBG out of the recommendations from NIST.
If elliptic curve DRBG is implemented, it is necessary to **not** follow the NIST generation process exactly due to the poor choice of truncation function[1]. Refer to Appendix B of [4] for more detail on TPP.
A few other cares must be taken before a DRBG is implemented. See Appendix C for a brief discussion on additional aspects to consider.

	Hash	HMAC	Block	EC
Security proof	✓	✓	✗	✓
Necessary underlying assumption(s)	Psuedo-randomness of the hash function	Psuedo-randomness of the hash function Resistance to key recovery	See section 6	CDHP, XLP, TPP
Sufficient underlying assumption(s)	Psuedo-randomness of the hash function	Psuedo-randomness of the hash function Resistance to key recovery	See section 6	DDHP, XLP, TPP
Maximum number of bits per request	2^{19}	2^{19}	2^{13} (with TDEA) 2^{19} (with all AES)	2^{32}
Maximum number of operations (generate)	2^{11} + 2 hash calls (128 bits security) 2^{10} + 2 hash calls (256 bits security)	$2^{11} + 2$ HMAC calls (128 bits security) $2^{10} + 2$ HMAC calls (256 bits security)	$2^7 + 4$ (with TDEA) 2^{12} + 3 (with AES-256)	$2^{33} + 1$ elliptic curve group scalar multiplications

[1]The truncation function that TPP addresses is not the truncation function t in the specification of [1]. TPP addresses the step when the algorithm concatenates the points (Step 8 of Dual_EC_DRBG Generate Process).

References

[1] Elaine Barker and John Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised). Technical report, NIST, 2007.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encrytion, 2000. `http://www-cse.ucsd.edu/users/mihir/crypto-research-papers.html`.

[3] Mihir Bellare. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. Cryptology ePrint Archive, Report 2006/043, 2006. `http://eprint.iacr.org/`.

[4] Daniel R. L. Brown and Kristian Gjøsteen. A Security Analysis of the NIST SP 800-90 Elliptic Curve Random Number Generator. Cryptology ePrint Archive, Report 2007/048, 2007. `http://eprint.iacr.org/`.

[5] Matthew J. Campagna. Security Bound for the NIST Codebook-Based Deterministic Random Number Generator. Cryptology ePrint Archive, Report 2006/379, 2006. `http://eprint.iacr.org/`.

[6] Scott Contini and Yiqun Lisa Yin. Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions. Cryptology ePrint Archive, Report 2006/319, 2006. `http://eprint.iacr.org/`.

[7] Yevgeniy Dodis, Rosario Gennaro, Johan Hastad, Hugo Krawczyk, and Tal Rabin. Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. *Advances in Cryptology - CRYPTO*, 2004.

[8] A. Muzereau, N.P. Smart, and F. Vercauteren. The Equivalence between the DHP and DLP for Elliptic Curves Used in Practical Applications. *London Mathematical Society*, 2004.

[9] Federal Information Processing Standards. The Keyed-Hash Message Authentication Code (HMAC). Technical report, NIST, 2002.

[10] Wikipedia. Cryptographically secure pseudorandom number generator — wikipedia, the free encyclopedia, 2007. [Online; accessed 20-July-2007].

[11] Wikipedia. Statistical randomness — wikipedia, the free encyclopedia, 2007. [Online; accessed 6-July-2007].

A Indistinguishability

Definition Let X and Y be two random variables. Let E be the following algorithm.
Inputs x, y be generated from X, Y respectively.
Generate $b \in_R \{0, 1\}$, where $\in_R$ denotes a uniform random choice.
If $b = 0$, output x,
otherwise, output y.
Two random variables X and Y are *computationally indistinguishable* if there is no polynomial-time algorithm to determine if an output u from E was generated from X or generated from Y with success rate better than $\frac{1}{2}$. We write $X \sim Y$.

Given $X \sim Y$, let A be any polynomial-time algorithm that tries to distinguish X and Y (i.e. upon receiving an input, A output either X if it decides the input came from X), then from the definition, one can see that $Prob(A \text{ is correct}) = Prob(A \text{ is incorrect}) = \frac{1}{2}$.
The following lemmas and proofs are adapted from [4].

Lemma 1. *'$\sim$' is an equivalent relation.*

Proof. '$\sim$' is clearly reflexive and symmetric.
Let X, Y, Z be three random variables such that $X \sim Y$ and $Y \sim Z$. Assume that there is an algorithm A such that it distinguishes X and Z.
We shall run A on inputs from X and Z.
Let $x = Prob(A \text{ outputs } X \mid \text{input is from } X)$,

$z = Prob(A \text{ outputs } Z \text{ |input is from } Z)$.
Then from the definition of indistinguishability, $x + z > 1$.
Now, we shall run A on an input from Y.
Let $x' = Prob(A \text{ outputs } X \text{ |input is from } Y)$,
$z' = Prob(A \text{ outputs } Z \text{ |input is from } Y)$.
Since A outputs either X or Z, $x' + z' = 1$.
Define $A_X(u) = \begin{cases} X, \text{ if } A(u) = X \\ Y, \text{ if } A(u) = Z \end{cases}$
Since $X \sim Y$, $x + z' = 1$.
Similarly, define $A_Z(u) = \begin{cases} Y, \text{ if } A(u) = X \\ Z, \text{ if } A(u) = Z \end{cases}$
and $x' + z = 1$. Thus, we have $x + z + x' + z' = 2$. Using $x' + z' = 1$, we get that $x + z = 1$, which contradicts the fact that $x + z > 1$. □

Lemma 2. *Let $f : X \cup Y \to Z$ be an efficient function. If $X \sim Y$, then $f(X) \sim f(Y)$.*

Proof. Assume that there is an algorithm A such that it distinguishes $f(X)$ and $f(Y)$. Define an algorithm B to distinguish between X and Y as followed:
Let u be the input for algorithm B
Apply f to input u
Apply A to $f(u)$
Output z, where z is the output of A to $f(u)$
Now we have a polynomial-time algorithm B to distinguish X and Y, since we assumed that f was efficient. Contradiction. □

The converse is not true. For example, it is believed that $(g^a, g^b, g^{ab}) \sim (g^a, g^b, g^c)$, where $a, b, c, g \in_R \mathbb{Z}/p\mathbb{Z}$ and all arithmetics are done in $\mathbb{Z}/p\mathbb{Z}$. However, (a, b, ab) is clearly distinguishable from (a, b, c) (Just multiply the first two terms.)
This lemma is trivial, but it has many useful consequences. For example, take $f(u) = (u, c)$, where c is some constant. This f shows that if $X \sim Y$, then $(X, constant) \sim (Y, constant)$. Take f as a hash function, then $X \sim Y$ implies that $hash(X) \sim hash(Y)$. Lastly, take f as a truncation function which removes the first k bits, then we can shorten two indistinguishable random variables and without changing the distinguishability.

Lemma 3. *If $X \sim Y$ and $W \sim Z$, such that X and W are independent and Y and Z are independent, then $(X, W) \sim (Y, Z)$.*

Proof.

Claim. *If $X \sim Y$, then $(X, Z) \sim (Y, Z)$.*

Proof of Claim. Assume that there is an algorithm A such that it distinguishes (X, Z) and (Y, Z). Define and algorithm B to distinguish between X and Y as followed:
Let u be the input for algorithm B
Select a random $z \in Z$
Apply A to (u, z)
Output q, where q is the output of A to (u, z)
Now we have an algorithm B to distinguish X and Y. Contradiction. □

Similarly, we can use the claim in a symmetric way to show that if $W \sim Z$, then $(X, W) \sim (X, Z)$. Thus, we have $(Y, Z) \sim (X, Z) \sim (X, W)$. By 1, we have $(Y, Z) \sim (X, W)$. □

The claim allows one to append or prepend indistinguishable random variables to indistinguishable random variable without changing its distinguishability.

B Proofs for Pseudo-random Output

Using the tools introduced in appendix A, we will proof that the output of hash DRBG and HMAC DRBG are pseudo-random under the assumption that the hash function along with a secret value used is a pseudo-random family.

Lemma 4. *Let b and k be independent bit strings uniformly distributed in $\{0,1\}^n$ and $\{0,1\}^l$ respectively. Let s be a bit string uniformly distributed in $\{0,1\}^m$. Define s_i inductively by $s_0 = s$ and $s_i = f(k, s_{i-1})$. If $f : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^n$ is a random oracle, then the following holds:*

For $l = 0$, s_0 the secret value: $(f(s_0+1), f(s_0+2), \ldots, f(s_0+i-1), f(s_0+i)) \sim (f(s_0+1), f(s_0+2), \ldots, f(s_0+i-1), b)$.

For $l \neq 0$, k the secret value: $(f(k,s_1), f(k,s_2), \ldots, f(k,s_{i-1}), f(k,s_i)) \sim (f(k,s_1), f(k,s_2), \ldots, f(k,s_{i-1}), b)$.

Proof. Follows from uniformity and independency. □

Proposition 1. *Let $b_1, b_2 \ldots, b_i$ be independent bit strings uniformly distributed in $\{0,1\}^n$. Let s be defined as in Lemma 4. If $f : \{0,1\}^m \to \{0,1\}^n$ is a pseudo-random family with the input as a secret value, then $(f(s+1), f(s+2), \ldots, f(s+i)) \sim (b_1, b_2, \ldots, b_i)$.*

Proof. Induction on i.
Case $i = 1$:

$$(f(s+1)) \sim (b_1) \quad \text{(Lemma 4)}$$

Case $i = j$:

$$\begin{aligned}
(f(s+1), f(s+2), \ldots, f(s+j-1)) &\sim (b_1, b_2, \ldots, b_{j-1}) && \text{(IH)}\\
(f(s+1), \ldots, f(s+j-1), f(s+j)) &\sim (f(s+1), \ldots, f(s+j-1), b_{j-1}) && \text{(Lemma 4)}\\
(f(s+1), \ldots, f(s+j-1), b_{j-1}) &\sim (b_1, b_2, \ldots, b_{j-1}, b_j) && \text{(Lemma 3)}\\
(f(s+1), \ldots, f(s+j-1), f(s+j)) &\sim (b_1, b_2, \ldots, b_{j-1}, b_j) && \text{(Lemma 1)}
\end{aligned}$$

□

Proposition 1 proves that the generation process for the hash DRBG outputs bits indistinguishable from truly random bits for a hash function that is a pseudo-random family with the input as a secret value.

Proposition 2. *Let $k, b_1, b_2 \ldots, b_i$ be independent bit strings uniformly distributed in $\{0,1\}^n$. Let s be defined as in Lemma 4. If $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ is a pseudo-random family with the first input as a secret value, then*

$$(f(k,s_1), f(k,s_2), \ldots, f(k,s_{i-1}), f(k,s_i)) \sim (b_1, b_2, \ldots, b_i).$$

Proof. Induction on i.
Case $i = 1$:

$$(f(k,s_1)) \sim (b_1) \quad \text{(Lemma 4)}$$

Case $i = j$:

$$\begin{aligned}
(f(k,s_1), f(k,s_2), \ldots, f(k,s_{j-1})) &\sim (b_1, b_2, \ldots, b_{j-1}) && \text{(IH)}\\
(f(k,s_1), f(k,s_2), \ldots, f(k,s_{j-1}), f(k,s_j)) &\sim (f(k,s_1), f(k,s_2), \ldots, f(k,s_{j-1}), b_j) && \text{(Lemma 4)}\\
(f(k,s_1), f(k,s_2), \ldots, f(k,s_{j-1}), b_j) &\sim (b_1, b_2, \ldots, b_{j-1}, b_j) && \text{(Lemma 3)}\\
(f(k,s_1), f(k,s_2), \ldots, f(k,s_{j-1}), f(k,s_j)) &\sim (b_1, b_2, \ldots, b_{j-1}, b_j) && \text{(Lemma 1)}
\end{aligned}$$

□

Proposition 2 proves that the generation process for the HMAC DRBG outputs bits is indistinguishable from truly random bits for an HMAC that is a pseudo-random family with the key as a secret value.

C Additional Notes on DRBGs

This article has only examined the output of a single call of the generate process. Two other ideas must be considered when one wishes to implement any of the DRBGs are: entropy extraction (the instantiation process) and the effect on the output of multiple calls of the generate process.

The instantiation process is the most vulnerable process for all the DRBGs. Any attacker that can find the secret values will be able to predict all future bits until reseeding (where reseeding is basically the instantiation process using states of the DRBG as inputs). The purpose of the instantiation process is to arrive at the secret values the DRBG needs from some entropy source.

Entropy extraction is a well studied area of cryptography. Many articles are dedicated to this matter. Specifically, in [7], generalized CBC, hash, HMAC based entropy extraction are analyzed under practical assumptions. Roughly speaking, for non-permutation based extraction, if security strength of k bits is needed, one should have $2k$ bits of entropy. For block cipher DRBG, the analysis is specifically for the entropy extraction using a derivation function and only k bits of entropy is needed. Ironically, the reasoning

for the need of fewer bits of entropy in block cipher is the same as why permutations are bad for truly random output; the lack of collisions ensures that no entropy is lost.
Multiple calls to the generate process is another area where the claimed security strength can fail. Imagine the following scenario to the HMAC based DRBG:
Suppose that there is an HMAC based DRBG always generates 2^k bits. Let V_i be the initial value of the secret value V. Let V_f be the value of the secret value V after the necessary number of calls to $HMAC(Key, V)$. For efficiency reason, *additional_input* is always 0x28. By chance, the call to $Update(0x28, Key, V_f)$ returns Key (the same key) and V_i. Thus, every output of this DRBG is identical. In another word, even if the HMAC is idealized, and a single output is indistinguishable from a true random source, the DRBG is completely predictable until reseeding.
However, this problem can be corrected simply by using a nonce (say time stamp) whenever additional inputs are required. Or use of the reseed counter as in hash based DRBG since that is incremented and changed until reseeding.