A Practice-Oriented Treatment of Pseudorandom Number Generators

Anand Desai[1], Alejandro Hevia[2], and Yiqun Lisa Yin[1]

[1] NTT Multimedia Communications Laboratories,
Palo Alto, California 94306, USA.
{desai,yiqun}@nttmcl.com

[2] University of California, San Diego,
La Jolla, California 92093, USA.
ahevia@cs.ucsd.edu

Abstract. We study Pseudorandom Number Generators (PRNGs) as used in practice. We first give a general security framework for PRNGs, incorporating the attacks that users are typically concerned about. We then analyze the most popular ones, including the ANSI X9.17 PRNG and the FIPS 186 PRNG. Our results also suggest ways in which these PRNGs can be made more efficient and more secure.

1 Introduction

Random numbers or bits are essential for virtually every cryptographic application. For example, seeds for key generation in both secret-key and public-key algorithms, session keys used for encryption and authentication, salts to be hashed with passwords, and challenges used in identification protocols are all assumed to be "random" by system designers. However, since generating enough randomness is expensive, most applications rely on a cryptographic mechanism, known as a Pseudorandom Number Generator (PRNG), to stretch a short string of random bits to a longer string of "random-looking" bits.

Background and Motivation. Cryptographic theory provides us with a number of constructions for PRNGs, such as the Blum-Blum-Shub generator [14] and the Blum-Micali generator [15], that are provably-secure under reasonable number-theoretic assumptions. However, for practical reasons, the PRNGs, that are in prevalent use today, are typically based on efficient cryptographic primitives such as block ciphers and hash functions. The two most widely-used PRNGs are the ANSI X9.17 PRNG [3] and the FIPS 186 PRNG [22]. The ANSI X9.17 PRNG is a part of a popular banking standard and was suggested (nearly the same time as DES) as a mechanism to generate DES keys and nonces. The FIPS 186 PRNG was standardized for generating randomness in DSA. These two constructions remain the only standardized PRNGs, despite there being many subsequent security-related standards. As a result, both of them are now being used as general-purpose PRNGs in various systems and applications.

There has been some analysis of the ANSI X9.17 PRNG and the FIPS 186 PRNG [29, 26], but it has been mostly ad hoc and based on heuristic arguments. There have been several theory-oriented treatments of PRNGs [34, 15] but these do not fully capture the way PRNGs are used in practice. Thus, despite their being around for some time now and their wide-spread usage, these PRNGs have yet to be validated in the tradition of provable security. Several cryptographic toolkits, both in commercial products and in free libraries, include general-purpose PRNG implementations that are different from the ANSI X9.17 and FIPS 186 PRNGs. None of these other PRNGs have been analyzed, in any rigorous sense, either.

OUR CONTRIBUTIONS. Our main goal is to study PRNGs as used in practice. We focus on the ANSI X9.17 and FIPS 186 PRNGs, not only because they are the most widely-used ones, but also because they represent the two typical design approaches for practical PRNGs– one is based on a keyed block cipher while the other is based on a keyless hash function. We also suggest ways of enhancing these PRNGs and look at some other practical ones.

A PRNG can be modeled as an iterative algorithm [1, 12]. Each iteration takes a single input called *state* (initial seed or intermediate state) and produces a random output of some fixed length and the next state. All the states are assumed to be hidden at all times. Although such a model seems sufficient for theoretical PRNGs, it does not capture all the nuances of a PRNG as used in practice. Indeed, a PRNG used in practice are often more complicated: (1) There are usually "auxilliary" inputs, such as time-stamps or counters, that the user (or even the attacker) may be able to control. (2) Some state information may be leaked out over time or modified by an attacker. (3) There is a wide range of cryptographic primitives on which they can be based, some that are secret-key based and others that are keyless.

We model a PRNG as an iterative algorithm, that in each iteration take *three* inputs: a key, a current state, and an auxiliary input. It generates two outputs: a PRNG output and a new state. "Pseudorandomness" roughly means that the output sequence should be "indistinguishable from a truly random sequence" to an attacker. To formalize the types of attacks on the above PRNG model, we look at the model from an attacker's point of view– the inputs can be hidden, known or chosen, while the outputs can be hidden or known. This leads to several different attacks, ranging from the "strongest" one where the attacker is allowed to choose all three inputs and see both the outputs to the "weakest" one where it does not see the key or the states and does not get to control the auxiliary input. Each attack coupled with the notion of "pseudorandomness" gives rise to a different definition of security. It is easy to see that there is not a strict hierarchy in the strengths of the attacks. Most of the attacks, that practical PRNGs resist, lie somewhere between the strongest and the weakest. We will be interested in the *strongest* attacks that a PRNG can be secure against since this gives us a complete picture of its security in our framework.

Our analysis of the ANSI X9.17 and the FIPS 186 PRNGs characterize the conditions on the inputs, states, and the underlying primitive that would ensure the security of the construction. They are also instructive as to what would or would not constitute secure-usage of these PRNGs. For the ANSI X9.17 we show that the construction is secure as long as the key is kept secret from the attacker and it is not allowed to control both the input and the state (though, neither needs to be kept secret from it). For the FIPS 186 PRNG we show that the construction is secure if the attacker cannot control the input. Unlike with the ANSI X9.17 PRNG, the key in this case is known to all, including the attacker. The states, however, must be kept secret. Besides validating the security of the PRNGs, our results have several useful practical implications. First, they suggest guidelines on how the PRNGs should be properly used to achieve their security objectives. Second, they suggest how to obtain some efficiency gains. For example, our results on the ANSI X9.17 PRNG imply that the construction can be made twice as efficient, by outputting intermediate states along with the output, without comprising security. Third, they suggest where the use of "good" randomness is critical and where it is not. For example, our results on the ANSI X9.17 PRNG, suggest that it is better to put all the randomness in the key rather than in the initial state. Finally, the analysis provides some insight on what security properties the underlying primitives should possess. This allows implementers to choose the appropriate primitives and to examine whether any newly-discovered weakness on the primitive can have a potential impact on the security of the PRNG.

A CLOSER LOOK. One important aspect of our framework is to allow for an auxiliary input to model things like time-stamps and counters. Auxiliary inputs are a common feature in practical PRNGs since they are a means of injecting something "random" into the PRNG at regular intervals and to prevent repeated seeds from causing repeated outputs. It is tempting to think that if a PRNG is secure without any auxiliary input then it must also be secure with it present. This, however, is not the case, particularly if this input can be controlled by the attacker. Indeed, in practice the auxiliary input may be supplied by a timer that an attacker may have some control over. Another aspect in which our model differs from most others is in making a distinction between the key and the state. When such a distinction is not made (and the key is simply understood to be a part of the state) then we miss out on being able to exactly characterize the security of the PRNG. The key and what we call as state play very different roles in a PRNG and moreover, they may have significantly different impacts on the overall security. Understanding this allows the user to make better use of the short seed it possesses.

A critical part of our analysis is to formalize reasonable security assumptions on the underlying primitives for the manner they are used in the ANSI X9.17 and the FIPS 186 PRNGs. There are no assumptions stated on the underlying primitive in the original ANSI X9.17 standard [3] or in the updated version (ANSI X9.31) [4]. In the FIPS 186 standard [22], it stated that the underlying primitive should be a "one-way" function. However, it is easy to see the one-way property itself is not enough to ensure the security of the construction. Going to an extreme, one may model the underlying primitive as a Random Oracle.

The proof of security, in this case, would indeed be straightforward, but there are some results [16] that cast doubt on whether this would be a reasonable assumption. Our assumption for both the constructions is that the underlying primitives are (finite) pseudorandom functions (PRFs). This is quite a natural assumption for the block-cipher-based ANSI X9.17 PRNG, following the work of Bellare, Kilian and Rogaway [10]. It is less so for the hash-function-based FIPS 186 PRNG, since there are no "secret keys" as such associated with the underlying function in this PRNG. However, we propose a different "view" of the PRNG under which it can be seen as based on a secret-keyed hash function. Under this view it seems reasonable to model the underlying function as a PRF. We note here that similar assumptions have been made before [9, 7, 6, 2] and that none of the known attacks on hash functions [18, 21, 32] suggest any weakness of their being used in this manner.

Starting with the strongest attacks in our framework, we look at the ability of the ANSI X9.17 and the FIPS 186 PRNGs to withstand them. For every attack stronger than the ones we eventually proved these PRNGS secure against, we identify the attack. Our approach helps us identify the criticality of each feature in these PRNGs. The superfluous ones can be eliminated leading to an improvement in the overall efficiency. It turns out that the designers of both these PRNGs anticipated many of the attacks in our framework and factored these into their designs– but not necessarily in the most efficient manner. We also suggest ways of making these PRNGs secure against attacks that, in their present form, they are insecure against. We remark that although the PRNGs we consider seem to incorporate many of the design features of better-known constructions, most notably of the CBC and Counter Mode constructs, their security analyses are quite different. This is partly due to the goal of the attacker being different and since the attacks we must consider are unlike any of those considered before. We show that if the primitives underlying the PRNGs are secure as PRFs then the PRNGs must be secure as well. Our analysis is *exact* rather than asymptotic.

Related Work. There have been numerous works on the theory of PRNGs, such as, Blum, Blum and Shub [14], Blum and Micali [15] and Hastad, Impagliazzo, Levin and Luby [27], to name just a few. However there have been comparatively few analyses of PRNGs of the type we study here, namely those based on efficient cryptographic primitives, such as block ciphers and hash functions.

Aeillo, Rajagopalan and Venkatesan [2] give a provably-secure design based on block ciphers. Bellare and Yee [12] and Abdalla and Bellare [1] also studied such PRNGs from the point of bringing forward-security to bear on them. They give generic methods of converting any PRNG into one that is forward-secure. We build on their security model to get a more general one.

While the above works did look at PRNGs from a provable-security viewpoint, there has not been anything similar done for the two most prevalent ones, namely the ANSI X9.17 and the FIPS 186 PRNGs. Kelsey, Schneier, Wagner and Hall [29] have, however, analysed these PRNGs, from an attackers viewpoint. That is, they give several different types of attacks and investigate the ability of these PRNGs to withstand them. While such analyses do not give the same level of guarantees as those we get, they are useful in identifying weaknesses. Indeed, many of the attacks on the PRNGs that become obvious in our framework were identified there as well. The only other work that we are aware of, which looks at one of these PRNGs, is the recent work of Bleichenbacher [13]. A weakness in the FIPS 186 PRNG, as used to generate DSA parameters, is identified. It does not apply to the general-purpose FIPS 186 PRNG we consider. The standard has anyway subsequently been revised [23] to function like the general-purpose one we consider.

There have been analyses of constructions bearing some similarity to the PRNGs we consider, namely the CBC MAC [10] and CBC and Counter Mode Encryption [8]. As mentioned earlier, the security of the PRNGs we consider do not follow from these analyses but the general provable-security techniques used therein are, however, applicable.

2 Security framework and definitions

PRNG Model. A PRNG $\mathcal{GE} = (\mathcal{K}, \mathcal{G})$ consists of two algorithms. The *seed generation* algorithm $\mathcal{K}$ takes as input a security parameter $k \in \mathsf{N}$ and returns a *key* K and an *initial state* s_0. For $i \geq 1$, the *generation* algorithm $\mathcal{G}$ takes as input the key K, the *current state* s_{i-1} and an *auxiliary input* t_i and returns a *PRNG output* y_i and the *next state* s_i. We refer to the length of the PRNG output in each iteration $n = |y_i|$ as the block length of the PRNG. Note that syntax requires the PRNG to have the "online" property. That is, it should be possible to generate y_i, s_i before t_{i+1} is known. We will sometimes refer to the auxiliary input simply as the "input" and the PRNG-output simply as the "output", when it is unambiguous from context.

We assume that there is a "good" source of randomness that accumulates in an *entropy pool* and that $\mathcal{K}$ has access to it. The process by which randomness is collected into the pool can be considered as orthogonal to the above-defined pseudorandom number generation process, and will not be considered further in our analysis. A couple of practical PRNGs [31, 28] do specify both processes as part of the PRNG definition. It is quite easy, however, to separate the two and analyze them independently.

Attack Models. Consider the normative operation of a PRNG as described above. At the start, a seed consisting of a key K and an initial state s_0 is generated from the entropy pool. After seed generation, the operation of the PRNG can be viewed as an application-controlled process— outputs are generated only as many as are required, using the *current* state and an auxiliary input. The auxiliary input may be a time-stamp, a counter or samples from a low entropy source. When the application stops the PRNG, the last state is usually computed and saved for use as the initial-state in the next invocation of the PRNG. The application may also choose to "re-seed" the PRNG at any time, if enough entropy has accumulated in the pool.

¿From an attacker's point of view, each of the three inputs to $\mathcal{G}$ can be hidden, known or chosen, while each of the two outputs from $\mathcal{G}$ can be hidden or known. Given the nature of a PRNG, it is not necessary to consider all the cases. We allow the *key* to be hidden or known, but not chosen. We view a key as defining the PRNG and hence a "chosen-key" attack would not help in "breaking" the PRNG being attacked as such. The *state* needs to be maintained between invocations of the PRNG. There is thus an increased probability that an attacker may learn of it. Moreover it may be able to "change" the value of the stored state. For these reasons we model attacks that allow the state to be hidden, known, or even chosen. The *auxiliary inputs* may not always be known a priori, even to a legitimate user. However, it is reasonable to assume that they are quite "predictable" by an attacker due to their low entropy. (Otherwise the generation of pseudorandom outputs would be trivial). Also, it is possible that the auxiliary inputs may be supplied by a timer or source over which an attacker may exercise some control. So we model the auxiliary inputs to be known or chosen. The *PRNG output* could conceivably be hidden, particularly if the PRNG is being used as a key-derivation function. However, since our focus is on general-purpose PRNGs, we will assume that the outputs become known. We further note that the choice of the primitive to realize the PRNG often rules out additional cases. For example, for PRNGs based on unkeyed hash functions the "key" is understood to be known. For PRNGs based on block ciphers the key is typically required to be hidden, though it is also be possible that it is known. Given the above, one can see why it is advantageous to consider the state and key separately in a practical setting. In most PRNGs the role played by the key and what we call the state are quite different. The key typically has a much longer lifetime and may be repeatedly used for different invocations of the PRNG. The part that we call the state has a more transient nature, since it is usually updated during every iteration of the generation algorithm.

Security Definitions. We adopt a standard notation with respect to probabilistic algorithms and sets. If $A(\cdot, \cdot, \ldots)$ is any probabilistic algorithm then $a \leftarrow A(x_1, x_2, \ldots)$ denotes the experiment of running A on inputs $x_1, x_2, \ldots$ and letting a be the outcome, the probability being over the coins of A. Similarly, if A is a set then $a \stackrel{R}{\leftarrow} A$ denotes the experiment of selecting a point uniformly from A and assigning a this value.

"Pseudorandomness" roughly means that the outputs should be "indistinguishable from random" to an attacker. Each attack in our framework coupled with the notion of "pseudorandomness" gives rise to a different security definition. Here we give more precise definitions for three of the attacks. We denote these attacks as CIA, for Chosen-Input Attack, CSA, for Chosen-State Attack and KKA, for Known-Key Attack. Under CIA, the key is hidden, the states are known, but not chosen, and the auxiliary input may be chosen by the attacker. CSA is similar, except that the auxiliary inputs are not allowed to be chosen while the states may now be chosen. KKA is different in that it allows the key to be known. However, under KKA, the states are hidden and the auxiliary inputs are not allowed to be chosen.

We imagine a distinguisher D running in two stages, an iterated find stage and a guess stage. In each iteration of the find stage D may get to choose some of the inputs (depending on the attack being modeled) that will be used to generate the PRNG output. In cases modeling a "chosen" input D's choice "overwrites" the existing value of that input. Depending on the bit b, D either gets the true PRNG output from the previous iteration or a random string. The find stage is iterated i times until $i = m$ or until D sets a flag p_i, indicating that it is ready to move on to the next stage. In the guess stage D receives some state information c_i gathered in the find stage and outputs its guess for b.

Definition 1. [PRG-CIA, PRG-CSA, PRG-KKA] *Let $\mathcal{GE} = (\mathcal{K}, \mathcal{G})$ be a PRNG with block-length n. (For simplicity, assume that the security parameter $k = n$.) Let $b \in \{0,1\}$. Let D be a distinguisher that runs in two stages. For* $\mathrm{atk} \in \{\mathrm{cia}, \mathrm{csa}, \mathrm{kka}\}$*, we consider the following experiment:*

Experiment $\mathbf{Exp}^{\text{prg-atk-}b}_{\mathcal{GE},m,D}(k)$
  $(K, s_0) \xleftarrow{R} \mathcal{K}(k)$ *// A key and an initial state are generated*
  $c_0 \leftarrow \{t_1, t_2, \ldots, t_m\}$ *// D is initialized with auxiliary input values*
  for $i = 1$ *to* $m :\ y_i^0 \xleftarrow{R} \{0,1\}^n$ *// A random mn-bit string is chosen*
  $i \leftarrow 0;\ y_0^0 \leftarrow \epsilon;\ y_0^1 \leftarrow \epsilon$
  repeat
    $i \leftarrow i + 1$
    if $\mathrm{atk} = \mathrm{cia} :\quad (p_i, t_i, c_i) \leftarrow D(\mathsf{find}, y^b_{i-1}, s_{i-1}, c_{i-1})$
    if $\mathrm{atk} = \mathrm{csa} :\quad (p_i, s_{i-1}, c_i) \leftarrow D(\mathsf{find}, y^b_{i-1}, s_{i-1}, c_{i-1})$
    if $\mathrm{atk} = \mathrm{kka} :\quad (p_i, c_i) \leftarrow D(\mathsf{find}, K, y^b_{i-1}, c_{i-1})$
    $(y_i^1, s_i) \leftarrow \mathcal{G}_K(s_{i-1}, t_i)$ *// PRNG output and next state are generated*
  until $(p_i = \mathsf{guess})$ *or* $(i = m)$
  $d \leftarrow D(\mathsf{guess}, c_i)$
  return d

We define the advantage of the distinguisher via

$$\mathbf{Adv}^{\text{prg-atk}}_{\mathcal{GE},m,D} = \Pr[\,\mathbf{Exp}^{\text{prg-atk-1}}_{\mathcal{GE},m,D} = 1\,] - \Pr[\,\mathbf{Exp}^{\text{prg-atk-0}}_{\mathcal{GE},m,D} = 1\,]\ .$$

We define the advantage function of $\mathcal{GE}$ as follows. For any integer t,

$$\mathbf{Adv}^{\text{prg-atk}}_{\mathcal{GE},m}(t) = \max_D\{\mathbf{Adv}^{\text{prg-atk}}_{\mathcal{GE},m,D}\}$$

where the maximum is over all D with "time complexity" t. ▮

The "time complexity" is the worst case total execution time of the experiment, plus the size of the code of the distinguisher, in some fixed RAM model of computation.

Note that the notions of security captured here are very strong but not the strongest in our framework. The reason for explicitly defining these and not the other possible ones was to give only as many definitions as were needed to give the complete picture about the security of PRNGs we will consider in this work. However, using the above as examples, it should be easy to come up with definitions modeling any attack in our framework.

It is instructive to see where these notions stand in our framework. When the key is hidden, the strongest notion would be one against a chosen-state, chosen-input and known-next-state attack. The next strongest notions, when the key is hidden, are PRG-CIA and PRG-CSA. When the key is known, the states must be hidden (otherwise there is no secret). This leaves us with only two notions: one allowing the input to be chosen and the other only allowing it to be known. PRG-KKA is the latter notion (which is the weaker of the two). Note that these three notions are mutually incomparable (ie. they are neither stronger nor weaker than one another).

3 Analysis of the ANSI X9.17 PRNG

Specifications. The ANSI X9.17 PRNG $\mathcal{GE}^F_{\text{ANSI}} = (\mathcal{K}_{\text{ANSI}}, \mathcal{G}_{\text{ANSI}})$, for a block cipher F, is described as in Figure 1. The key K, output by the key-generation algorithm, is used to key the block cipher, thereby specifying a function F_K that maps n bits to n bits.

$\underline{\mathcal{K}_{\text{ANSI}}(n)}$	$\underline{\mathcal{G}_{\text{ANSI}}(s_{i-1}, t_i)}$
$K \stackrel{R}{\leftarrow} \{0,1\}^n$	$y_i \leftarrow F_K(s_{i-1} \oplus F_K(t_i))$
$s_0 \stackrel{R}{\leftarrow} \{0,1\}^n$	$s_i \leftarrow F_K(y_i \oplus F_K(t_i))$
return (K, s_0)	return (y_i, s_i)

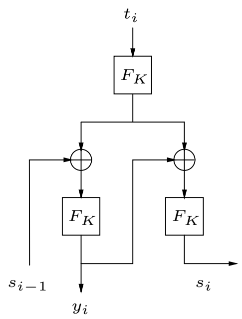

Fig. 1. Specifications of the ANSI X9.17 PRNG. The picture depicts the function $\mathcal{G}_{\text{ANSI}}(\cdot, \cdot)$.

Attacks. The PRNG is insecure under any attack in which the key is known. An attacker, knowing the key and a single output, can completely determine subsequent outputs and states. The PRNG is also not secure under any attack where both the state and the inputs may be chosen. It is easy to see that an attacker can cause outputs to repeat by simply repeating the same combination of the state and input. Given this, we focus on the setting where the key is hidden and where either the states or the inputs are not under the control of the attacker.

Finite PRF families. We model block ciphers as finite pseudorandom function families [10], a concrete security version of the original notion of pseudorandom functions [25]. A finite function family F is pseudorandom if, for a random K, the input-output behavior of F_K is indistinguishable from that of a random function of the same domain and range. Following [10], we associate an advantage function $\mathbf{Adv}_F^{\mathrm{prf}}(t,q)$ with F which denotes the maximum advantage of any distinguisher, with time complexity t and query complexity q. The (standard) formalization of this can be found in the full version of this paper [19].

Security Results. The following theorem implies that, if F is a PRF, then the ANSI X9.17 PRNG based on it is secure in the PRG-CSA sense and also in the PRG-CIA sense. Note that the two notions are mutually incomparable and thus proving the PRNG secure under one does not imply security under the other.

Theorem 1. *Let $\mathcal{GE}$ be the ANSI X9.17 PRNG based on a function family $F = \{F_K\}_{K\in\mathrm{Keys}(F)}$ where F_K maps n bits to n bits. Then*

$$\mathbf{Adv}_{\mathcal{GE},m}^{\text{prg-csa}}(t) \le 2\cdot \mathbf{Adv}_F^{\mathrm{prf}}(t,3m) + \frac{m(2m-1)}{2^n}$$

$$\mathbf{Adv}_{\mathcal{GE},m}^{\text{prg-cia}}(t) \le 2\cdot \mathbf{Adv}_F^{\mathrm{prf}}(t,3m) + \frac{(2m-1)^2}{2^n} \quad \blacksquare$$

A proof of the theorem is given in the full version of this paper [19]. If we model F as a PRP (a more suitable model for a block cipher) then an additional term $(3m)^2 \cdot 2^{-n-1}$ would appear in the bounds above. The proofs of security under the two notions share much in common. We first analyze the PRNG, under each notion, assuming that the underlying function is truly random. In both cases we show that, with overwhelming probability, an attacker cannot cause collisions in the inputs to the functions computing the outputs or the next states. Under CSA it is the inputs, which essentially function as a counter, that make it infeasible to cause these collisions. Under CIA, it is the unpredictability of the states that makes it infeasible. It is easy to see that, if there are no such collisions, then the attacker has no advantage in distinguishing between PRNG outputs and random ones. The next part of the analysis is a reduction argument that shows that, if the random function in the above analysis were replaced by a PRF, then the PRNG must remain secure.

Remarks. Our results suggest how the PRNG could be used more efficiently without compromising security. Notice that in the analysis, under either of the notions, we assume that the attacker learns of the states. Moreover, the states themselves are indistinguishable from random to the attacker. (In fact, there is complete symmetry in the way the outputs and the states are computed.) This means that the states could be used as "outputs", effectively doubling the throughput of the PRNG, without any loss in security. Our analysis also offers some insights into how the PRNG can be extended to counter the attacks that it does not in its present form. We discuss this further in Section 5.

4 Analysis of the FIPS 186 PRNG

Specifications. The FIPS 186 PRNG $\mathcal{GE}_{\mathrm{FIPS}}^H = (\mathcal{K}_{\mathrm{FIPS}}, \mathcal{G}_{\mathrm{FIPS}})$, for a function H, is described as in Figure 2.

$\underline{\mathcal{K}_{\text{FIPS}}(n)}$	$\underline{\mathcal{G}_{\text{FIPS}}(s_{i-1}, t_i)}$
$K \leftarrow \{0,1\}^n$	$y_i \leftarrow H_K((s_{i-1} + t_i) \bmod 2^n)$
$s_0 \xleftarrow{R} \{0,1\}^n$	$s_i \leftarrow (s_{i-1} + y_i + 1) \bmod 2^n$
return (K, s_0)	return (y_i, s_i)

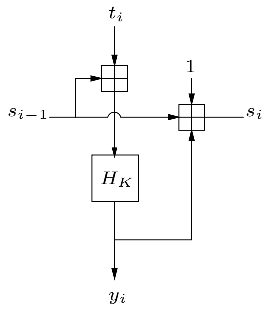

Fig. 2. Specifications of the FIPS PRNG. The picture depicts the function $\mathcal{G}_{\text{FIPS}}(\cdot, \cdot)$.

Note that in this PRNG, unlike in the ANSI X9.17 one, the key K is *known*, and the values for K are *fixed* in the original specifications [22]. [1] To fully specify this PRNG, we must describe the underlying primitive H_K. However, for the sake of simplicity of exposition, we defer this to the end of this section, viewing H_K, for now, simply as a function mapping n bits to n bits.

Attacks. Since the key K is known, the only secret in the PRNG is the initial state s_0. Clearly, if s_0 is known to the attacker then the outputs become completely predictable. In general, such a PRNG cannot be secure in any attack where the state is known. It also turns out that this PRNG is not secure under any attack where the inputs are chosen since otherwise it becomes possible for an attacker to cause outputs to repeat. Given this, we focus on the setting where the states are hidden and the inputs are not under the control of an attacker.

An Alternative View. We first need to formalize a security assumption on the underlying primitive H_K that is reasonable for the manner in which it is realized and used in the specifications. Towards this, we will consider an alternative view of FIPS. We start with some notation. Let

$$\hat{H}_{s_0}(x) \stackrel{\text{def}}{=} H_K((s_0 + x) \bmod 2^n).$$

For simplicity of notation we do not have an explicit reference to K in the definiton of $\hat{H}_{s_0}$. Nevertheless, one should keep in mind that K is a part of the definition of $\hat{H}_{s_0}$. Now, we can rewrite the specifications of the FIPS 186 PRNG as in Figure 3.

[1] In the original FIPS specifications [22], the underlying primitive is denoted by $G(K, x)$, where K is a constant and x is the input. It specifies two 160-bit values for the constant K, with each constant instantiating a different PRNG process for different use in DSA. Although the specifications do not interpret K as a key, the construction fits naturally into our PRNG model if we view K as a known key. The FIPS 186 construction can be generalized to have an arbitrary 160-bit value (even secret) for K.

$\mathcal{K}_{\text{FIPS}}(n)$	$\hat{\mathcal{G}}_{\text{FIPS}}(s_{i-1}, t_i)$
$K \leftarrow \{0,1\}^n$	$\Delta s_{i-1} \leftarrow (s_{i-1} - s_0) \bmod 2^n$
$s_0 \xleftarrow{R} \{0,1\}^n$	$y_i \leftarrow \hat{H}_{s_0}((\Delta s_{i-1} + t_i) \bmod 2^n)$
	$s_i \leftarrow (s_{i-1} + y_i + 1) \bmod 2^n$
return (K, s_0)	return (y_i, s_i)

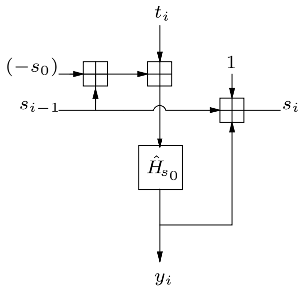

Fig. 3. Alternative specifications of the FIPS 186 PRNG. The picture depicts the function $\hat{\mathcal{G}}_{\text{FIPS}}(\cdot,\cdot)$, which is an equivalent view of $\mathcal{G}_{\text{FIPS}}(\cdot,\cdot)$.

Note that the alternative specifications are simply another way of viewing the original FIPS 186 PRNG. The two specifications interpreted literally will have different-looking implementations but they will, nevertheless, have exactly the same input-output characteristics. We view it in our "secret-key" function-based form for sake of analysis.

Security Results. The following theorem implies that, if $\hat{H}$ is a PRF family, then the FIPS 186 PRNG based on it is secure in the PRG-KKA sense.

Theorem 2. *Let $\mathcal{GE}$ be the FIPS 186 PRNG based on a function family $\hat{H} = \{\hat{H}_{s_0}\}_{s_0 \in \{0,1\}^n}$. Then*

$$\mathbf{Adv}^{\text{prg-kka}}_{\mathcal{GE},m}(t) \leq 2 \cdot \mathbf{Adv}^{\text{prf}}_{\hat{H}}(t,m) + \frac{m(m-1)}{2^{n-1}} \quad \blacksquare$$

A proof of the theorem is given in the full version of this paper [19]. The approach in the analysis is much like the one for the ANSI X9.17 PRNG, where we first analyze the PRNG assuming the underlying function to be random and then give a reduction argument to show that if the PRNG is secure with the random function then it must also be secure with a PRF. The interesting part for this particular analysis is in the reduction argument. Recall that for this argument, we must construct a distinguisher A for the PRF that simulates the PRNG experiment by calling its oracle f on the input $(\Delta s_{i-1} + t_i) \bmod 2^n$. It is easy to see that $\Delta s_i = (\Delta s_{i-1} + y_i + 1) \bmod 2^n$ can be computed without knowledge of s_0. This makes it possible for A to make the necessary queries and the argument goes through.

The function H_K and the function family $\hat{H}$. It remains to give some details about the function H_K and to comment on the security of the function family $\hat{H}$. The FIPS 186 specifications provides two constructions for the function H_K, one based on SHA1 and the other based on DES.

The SHA1-based construction is fairly simple— $H_K(x)$ is defined to be the compression function of SHA1 where x is the input to the function and K is the fixed IV in SHA1. Our assumption is that one can view $\hat{H}_{s_0}(x) = H_K((s_0 + x) \bmod 2^n)$ for a known K and a secret random s_0 of length n, as a PRF. As mentioned earlier, this assumption is not entirely new. Bellare et al [9, 7, 6] make an assumption that, in our context, amounts to viewing $H_K(s_0, x)$ as a PRF. It appears as though all existing attacks on hash functions need to either fix certain bits of the input to the compression function or exploit the underlying iterative structure of the functions. Since neither of these seem possible in our setting, we believe that it is reasonable to view the compression function of SHA1, as used in the FIPS 186 PRNG, as a PRF.

The DES-based construction is more involved— DES encryption is performed five times with a different secret key and input each time. The secret keys and the inputs are both derived from K and x. It is significantly more complicated to gauge whether a DES-based $\hat{H}$ function family can be considered to be a PRF family. Anyway, it appears that the DES-based construction is rarely, if at all, used in practice.

5 Forward security for PRNGs

The notion of forward security has been applied to a range of cryptographic problems. It is a particularly desirable goal for PRNGs. Indeed, the idea itself seems to have originated in the PRNG context. Informally, a PRNG is said to be forward secure if the compromise of the current state *and* key does not compromise the security of any previously generated PRNG-output. Two different formulations of the notion of forward security in PRNGs have appeared in the literature. We will look at each in turn.

The first, that we will refer to as *weak* forward security, assumes that the PRNG-outputs, prior to the compromise, are *hidden* from the attacker. The goal of the attacker is to derive information about the earlier outputs. This notion seems to have been considered by Kelsey et al [29]. Although the term "forward security" was not explicitly used, the "backtracking attack" introduced therein is precisely the type of attack implicit in the notion of weak forward security.

A more comprehensive study of the notion of forward security in the context of PRNGs was initiated in the works of Bellare and Yee [12] and Abdalla and Bellare [1]. They considered a stronger formulation of the notion, that we will refer to as *strong* forward security. In this notion it is assumed that the outputs prior to the compromise are *known* to the attacker. The goal of the attacker is to distinguish the earlier output sequence from a truly random one.

Bellare and Yee [12] suggest ways of making a strong forward-secure PRNG out of any generic PRF-based PRNG. The main idea there is to keep part of the PRNG output secret and use it to get a new state and key. The idea applied quite literally to the PRNGs we consider result in unnecessarily inefficient constructions. Moreover, the fact that their analysis did not consider auxilliary inputs or make a distinction between the key and the state, makes it somewhat difficult to see if their method would be secure in our setting. Nevertheless, we believe that one could use their basic idea involving re-keying to get forward security for both the PRNGs we consider.

We now look at some alternate approaches to this problem, specifically for the ANSI X9.17 and the FIPS 186 PRNGs. We want to avoid, as far as possible, necessitating major changes to the constructions (such as using a different underlying primitive), so that our "fixes" can be applied to existing implementations of the PRNGs. For efficiency reasons we look for solutions that do not involve re-keying the underlying functions.

The Case of the ANSI X9.17 PRNG. It is easy to see that this PRNG is not forward secure, under even the weaker form, since revealing the (secret) key would make the PRNG process completely reversible. As a first step towards bringing forward security to bear upon this PRNG, we must transform the underlying function F_K into a non-invertible function (even when the key K is known). While such a change may be sufficient for weak forward security, a more subtle attack implies that the modified version will still be insecure under the stronger notion. The attack is to check, using the last output, time-stamp and key, if the state-update function does in fact give the current state.

The above attack suggests that a way to get strong forward security is to require that the state-update function use a value which remains *secret* even when the key and the current state are compromised (in addition to using a non-invertible function). One possibility is to change the state-update function from $s_i \leftarrow F_K(y_i \oplus F_K(t_i))$ to $s_i \leftarrow F_K(y_i \oplus s_{i-1})$. It remains to discuss how we can get a non-invertible function. What we need specifically is a function F', such that given $F'_K(x)$ and K, it is infeasible to get information about x. One can build such a function using ideas from converting PRPs to PRFs [11]. However these require rekeying. One possible candidate that avoids this and may suffice for our purposes is the function F' defined as $F'_K(x) = F_K(x) \oplus x$, where F is a block cipher.

The Case of the FIPS 186 PRNG. Observe that the "feedback" structure (ie. using output y_i as an input to the state-update function) in this PRNG does not really play a role in preventing against any of the attacks considered in our framework. However, it is precisely this feature that makes this PRNG weak forward-secure. The PRNG, however, is completely insecure under the stronger notion. Given the current state and the outputs, every previous state, going back to the initial state can be determined. Note that the standard attacks on this PRNG anyway make it necessary for the output-computation function to be "perfectly" one-way. Thus if we want to make it strong forward secure, then we only need to modify the state-update function. One possibility is to use a hash function in the state update. (The function H_K used in the PRNG may be used as this hash function.)

6 Other practical PRNGs

We have, so far, concentrated on just the ANSI X9.17 and the FIPS 186 PRNGs. Other than being the two most popular PRNGs, they are also representative of the two main types– secret-keyed block-cipher-based ones and keyless hash-function-based ones. Most other practical PRNGs are similar to one of these and it is relatively straightforward to analyze them using the same framework and approach. We discuss some of these below.

PRNGs in Standards. We have reviewed the PRNGs in most security-related standards of ANSI, FIPS, IETF, IEEE and ISO. Not surprisingly, the ANSI X9.17 and FIPS 186 PRNGs have been chosen by subsequent ANSI and FIPS standards. In addition, both PRNGs have been re-validated and cross-validated by the two standards bodies (e.g., ANSI X9.31 [4] and FIPS 186-2 (change notice 1) [23]). Most other standards do not specify any concrete PRNG construction but merely provide guidelines on how random numbers should be generated. The ANSI X9.17 and FIPS 186 constructions are, as far as we know, the only standardized PRNGs.

The only relevant PRNG-like construction that has appeared in standards is the $\mathtt{PRF}(k, seed)$ function in the Transport Layer Security Protocol (TLS) [20] (TLS is part of the IETF effort to standardize SSL [24].) Despite its name, `PRF` is used for key derivation rather than generating random numbers needed in the protocol. The latter was assumed to be generated by some "secure PRNG." Nevertheless, `PRF` is of interest to us since it does represent a standalone pseudorandom number generating process. At a high level, `PRF` is a secret-key based construction using HMAC [6] as the underlying primitive. The initial state is assumed a "random" but public value, and there is no user-supplied input. HMAC is used to both generate the output and update the state. Our preliminary analysis suggests that `PRF` is secure under known-state attacks, but not secure under chosen-state attacks. We remark that this apparent weakness does not have any immediate impact on the security of TLS itself.

PRNGs in Commercial Products and Free Libraries. There are several cryptographic toolkits, both in commercial products and in free libraries, which are likely to be used when implementing cryptographic solutions. Among commercial products, NAI's PGP [31] and RSA's BSAFE [5] are perhaps the two most widely-used ones. Among popular free libraries, we considered CryptoLib [30], OpenSSL [35], Crypto++ [17], RSAREF [33] and Yarrow [28], all of which include general-purpose PRNG implementations.

Recently NAI released the source code for PGPsdk [31] for peer review. This version of PGP specifies the default PRNG to be ANSI X9.17 with the block cipher CAST5. It also implements a quite complicated process for generating the initial seed and time-stamps required by the ANSI X9.17 PRNG. There seems to be some uncertainty as to whether the ANSI X9.17 PRNG can produce pseudorandom output if the seed and time-stamps do not have enough entropy. Our analysis, however, shows that the PRNG is secure as long as the key is secret and random and the attacker does not have control over both the state and time-stamps. Thus, it seems that the complication involved in generating pseudorandom inputs in PGP may be unnecessary.

The PRNG included in the BSAFE toolkit [5] is a hash-function-based construction with no secret key and with user-inputs processed as they become available. The construction bears some resemblence to the forward-secure variant of the FIPS PRNG we suggest in Section 5, without the output feedback.

Our preliminary analysis shows that the BSAFE PRNG has similar security properties as the FIPS PRNG. That is, it is secure under known-input attacks but not secure under chosen-input attacks. The BSAFE PRNG also has the additional feature of allowing multiple output-computation between two state updates.

The Crypto++ library includes a PRNG based on a (less efficient) variant of the ANSI X9.17 construction. The PRNG implemented in OpenSSL bears some similarities with the FIPS 186 construction but is a lot more complicated and is less efficient. The PRNGs in CryptoLib, RSAREF and Yarrow, on the other hand, have a quite simple Counter-Mode structure. CryptoLib's PRNG implements a scheme that combines the encryption and hash of a counter. The RSAREF PRNG uses a hash function to compute digests of the counter and inputs. Poor handling of inputs, however, have raised concerns about its security [29]. The Yarrow PRNG seems to be a carefully-designed secret-key-based construction using Counter-Mode encryption. It provides some level of forward security by performing frequent (hash-based) re-seeding and re-keying operations.

Acknowledgements

We thank Mihir Bellare and the members of the Program Committee for their helpful comments.

References

1. M. Abdalla and M. Bellare, "Increasing the Lifetime of a Key: A Comparative Analysis of the Security of Re-keying Techniques," ASIACRYPT 2000.
2. W. Aiello, S. Rajagopalan, and R. Venkatsan, "High-Speed Pseudorandom Number Generation with Small Memory," FSE 1999.
3. ANSI X9.17 (Revised), "American National Standard for Financial Institution Key Management (Wholesale)," America Bankers Association, 1985.
4. ANSI X9.31, "American National Standard for Financial Institution Key Management (Wholesale)," America Bankers Association, 2001.
5. R. Baldwin, "Preliminary Analysis of the BSAFE 3.x Pseudorandom Number Generators," *RSA Laboratories*' Bulletin No. 8, 1998.
6. M. Bellare, R. Canetti and H.Krawczyk, "Keying Hash Functions for Message Authentication," CRYPTO 1996.
7. M. Bellare, R. Canetti and H. Krawczyk, "Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security," FOCS 1996.
8. M. Bellare, A. Desai, E. Jokipii and P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption," FOCS 1997.
9. M. Bellare, R. Guérin, and P. Rogaway, "XOR MACs: New Methods for Message Authentication using Finite Pseudorandom Functions," CRYPTO 1995.
10. M. Bellare, J. Kilian and P. Rogaway, "The Security of the Cipher Block Chaining Message Authentication Code," CRYPTO 1994.
11. M. Bellare, T. Krovetz, and P. Rogaway, "Luby-Rackoff Backwards: Increasing Security by making Block Ciphers Non-Invertible," EUROCRYPT 1998.

12. M. Bellare and B. Yee, "Forward Security in Private-Key Cryptography," *Cryptology ePrint Archive*, Report 2001/035.
13. D. Bleichenbacher, Lucent Technologies Press Release, `http://www.lucent.com/press/0201/010205.bla.html`.
14. L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudorandom Number Generator." *SIAM J. Computing*, 15(2), 1986.
15. M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudorandom Bits." *SIAM J. Computing*, 13, 1984.
16. R. Canetti, O. Goldreich, and S. Halevi, "The Random Oracle Model, Revisited." STOC 1998.
17. W. Dai, Crypto++ Library. `http://www.eskimo.com/~weidai/cryptlib.html`
18. B. den Boer and A. Bosselaers, "Collisions for the Compression Function of MD5," CRYPTO 1993.
19. A. Desai, A. Hevia and Y.L. Yin, "A Practice-Oriented Treatment of Pseudorandom Number Generators," `http://www.cs.ucsd.edu/users/adesai`.
20. T. Dierks, and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, *Internet Request for Comments*, 1999.
21. H. Dobbertin, "The Status of MD5 After a Recent Attack," RSA Labs' CryptoBytes, Vol.2 No.2, 1996.
22. FIPS PUB 186-2, "Digital Signature Standard," National Institute of Standards and Technologies, 1994.
23. FIPS PUB 186-2 (Change Notice 1), "Digital Signature Standard," National Institute of Standards and Technologies, 2001.
24. A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corp., Nov 18, 1996.
25. O. Goldreich, S. Goldwasser, and S. Micali, "How to Construct Random Functions," *Journal of the ACM*, Vol. 33, N0. 4, 1986.
26. P. Gutmann, "Software Generation of Practically Strong Random Numbers," USENIX Security Symposium 1998.
27. J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby, "Pseudorandom Generation from One-Way Functions," STOC 1989.
28. J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator," SAC 1999.
29. J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic Attacks on Pseudorandom Number Generators," FSE 1998.
30. J.B. Lacy, D.P. Mitchell, and V.M. Schell, "Cryptolib: Cryptography in Software," USENIX Security Symposium, 1993.
31. Network Associates, Inc., "PGPsdk 2.1.1 Source Code for Peer Review".
32. P. van Oorschot and M. Wiener "Parallel Collision Search with Applications to Hash Functions and Discrete Logarithms," ACM CCS 1994.
33. RSA Laboratories, "RSAREF Cryptographic Library," 1994.
34. A.C. Yao, "Theory and Applications of Trapdoor Functions," FOCS 1982.
35. E.A. Young and T.J. Hudson, "OpenSSL Library v.0.9.6c," 2001.