SoK: Security Models for Pseudo-Random Number Generators

Sylvain Ruhault

March 8th, Tokyo, Fast Software Encryption 2017
Motivation

Papers about PRNG

- FSE 96: Jenkins
- FSE 98: Schneier et al.
- Usenix 98: Gutman
- EC02: Desai et al.
- CT-RSA03: Bellare and Yee
- ACSAC03: Viega
- CHES03: Barak et al.
- CCS05: Barak and Halevi
- CCS10: Yu et al.
- CCS13: Dodis et al.
- C14: Dodis et al.
- ...

SoK Paper

- Unify security models presentation
- Propose secure constructions based on AES
Standard PRNG

Stateful PRNG

PRNG with input
Standard PRNG

Security of $G$
- Secret $S$
- $|R| > |S|$
- $R$ is indistinguishable from random.

AES based construction
- $S \leftarrow \{0, 1\}^{128}$
- $R = AES_S(1) || AES_S(2) || \cdots$
Stateful PRNG

- $R_0, R_1, \ldots$ shall be indistinguishable from random
- $S$: internal state of the generator
Stateful PRNG

- $R_0, R_1, \cdots$ shall be indistinguishable from random

- $S$: internal state of the generator
State Compromise

Forward Security

- Past outputs are not compromised
- Can be build upon a secure standard PRNG (BY03)
**State Compromise**

![Diagram showing state compromise]

**Forward Security**
- Past outputs are not compromised
- Can be build upon a secure standard PRNG (BY03)

**AES based construction**

key

Require: ∅  
Ensure: S

1: $S \leftarrow \{0,1\}^{128}$  
2: return $S$

next

Require: S  
Ensure: $S'$, $R$

1: $S' = \text{AES}_S(1)$  
2: $R = \text{AES}_S(2)$  
3: return $(S', R)$
State Compromise

Forward Security
- Past outputs are not compromised
- Can be build upon a secure standard PRNG (BY03)

Backward Security?
- "Next" outputs are not compromised?
  - New input shall be collected
  - Recovery mechanism
PRNG with input

\[ I \rightarrow G \rightarrow S \rightarrow R \]
PRNG with input

How to Manage Inputs?

- **Accumulation**: entropy of each input shall be accumulated in the internal state
- **Extraction**: entropy of the collected inputs shall be extracted to generate outputs

→ these operations are implicit in Fortuna, OpenSSL PRNG, /dev/random, NIST CTR_DRBG, ...
PRNG with input

How to Manage Inputs?

- **Accumulation**: entropy of each input shall be accumulated in the internal state
- **Extraction**: entropy of the collected inputs shall be extracted to generate outputs

⇝ these operations are implicit in Fortuna, OpenSSL PRNG, /dev/random, NIST CTR_DRBG, ...

Definitions

- **Seeded** extractors, accumulators
- Requires **independence** between public seed and inputs

⇝ Potential vulnerability in NIST CTR_DRBG
Barak-Halevi Model (BH05)

PRNG with input Definition

Two operations
- input collection
- output generation

Where
- Operations are not synchronised

\[ S \xrightarrow{\text{refresh}} S' \]

\[ S \xrightarrow{\text{next}} S' \]

\[ I \]

\[ R \]
Recovery in Barak-Halevi model
Recovery in Barak-Halevi model

AES based construction

setup
Require: $r$
Ensure: $X$
1: $X \leftarrow \{0, 1\}^{512}$
2: return $X$

refresh
Require: $X, I, S$
Ensure: $S'$
1: $U = [X \cdot I]_{128}$
2: $S' = S \oplus U$
3: return $S'$

next
Require: $S$
Ensure: $(S', R)$
1: $S' = \text{AES}_S(1)$
2: $R = \text{AES}_S(2)$
3: return $(S', R)$
Security Analysis

AES based construction

setup
Require: \( r \)
Ensure: \( X \)
1: \( X \leftarrow \{0, 1\}^{512} \)
2: return \( X \)

refresh
Require: \( X, I, S \)
Ensure: \( S' \)
1: \( U = [X \cdot I]_{128} \)
2: \( S' = S \oplus U \)
3: return \( S' \)

next
Require: \( S \)
Ensure: \( S', R \)
1: \( S' = \text{AES}_S(1) \)
2: \( R = \text{AES}_S(2) \)
3: return \( (S', R) \)

- \( |S| = 128 \)
- Involves a **Seeded Extractor**
- At least one input shall have maximal entropy \( H_\infty(I) = 512 \)
- Requires a public random seed \( X \) of length 512 bits
- Inputs shall be independent from \( X \)
Dodis et al. Model (DPR+13)

PRNG with input Definition

Triple of algorithms (setup, refresh, next):
- **setup**, seed generation algorithm
- **refresh**, entropy collecting algorithm, \((S, I) \rightarrow S'\)
- **next**, output algorithm, \(S \rightarrow (R, S')\)
Recovery in Dodis et al. Model

- entropy can be accumulated slowly in $S$
- recovery: after accumulated entropy is OK
Recovery in Dodis et al. Model

AES based construction

setup

Require: ∅
Ensure: X, X′

1: $X \leftarrow \{0, 1\}^{1024}$
2: $X′ \leftarrow \{0, 1\}^{1024}$
3: return $X, X′$

refresh

Require: X, I, S
Ensure: S′

1: $S′ = S \cdot X + I$
2: return $S′$

next

Require: S, X′
Ensure: S′, R

1: $U = [X′ \cdot S]_{256}$
2: $S′ = \text{AES}_U(1) \| \cdots \| \text{AES}_U(8)$
3: $R = \text{AES}_U(9)$
4: return $(S′, R)$
Security Analysis

AES based construction

setup
Require: \( \varnothing \)
Ensure: \( X, X' \)

1: \( X \leftarrow \{0, 1\}^{1024} \)
2: \( X' \leftarrow \{0, 1\}^{1024} \)
3: \text{return } X, X'

refresh
Require: \( X, I, S \)
Ensure: \( S' \)

1: \( S' = S \cdot X + I \)
2: \text{return } S'

next
Require: \( S, X' \)
Ensure: \( S', R \)

1: \( U = [X' \cdot S]_{256} \)
2: \( S' = AES_U(1) || \cdots || AES_U(8) \)
3: \( R = AES_U(9) \)
4: \text{return } (S', R)

- \( |S| = 1024 \)
- Involves a **Seeded Extractor** and a **Seeded Accumulator**
- Requires a public random \( (X, X') \) of length 2048 bits
- Inputs shall be independent from \( X \)
- Extensions has been proposed for Leakage Security [CR14, ABPRV15]
Premature Next Attack, Dodis et al. (DSSW14)

- a next call can be done **before** recovery
Premature Next Attack, Dodis et al. (DSSW14)

\[ S \rightarrow \text{next} \rightarrow S \rightarrow \text{next} \rightarrow S \rightarrow \text{next} \rightarrow S \rightarrow \ldots \]

- a next call can be done before recovery

Solution: \( S = [S_1 \cdots S_{in} \cdots S_{out} \cdots S_p] \), a scheduler selects \( S_{in} \) and \( S_{out} \)
Generalized Fortuna Construction (DSSW14)

\( G_i, i = 1, \cdots, 32, \) based on AES

AES based scheduler

- Uses AES as a PRF
- \((\text{in, out}) \leftarrow SC(skey)\)

AES based construction
Security Analysis

AES based construction

**setup**

Require:  \( \emptyset \)

Ensure:  \( X, X', skey \)

1: \( X, X' \leftarrow \text{setup}_G \)
2: \( skey \leftarrow \{0,1\}^{128} \)
3: \( \text{return } X, X', skey \)

**refresh**

Require:  \( X, key, l, S \)

Ensure:  \( S' \)

1: \( \text{parse } S \text{ as } (S_\rho, (S_i)_{i=0}^{31}) \)
2: \( (in, out) \leftarrow SC(skey) \)
3: \( S_{in} \leftarrow \text{refresh}_{in}(X, S_{in}, l) \)
4: \( (S_{out}, R) \leftarrow \text{next}_{out}(X', S_{out}) \)
5: \( S_\rho \leftarrow S_\rho \oplus R \)
6: \( \text{return } S' = (S_\rho, (S_i)_{i=0}^{31}) \)

**next**

Require:  \( S \)

Ensure:  \( S', R \)

1: \( \text{parse } S \text{ as } (S_\rho, (S_i)_{i=0}^{31}) \)
2: \( S_\rho = \text{AES}_{S_\rho}(1) || \text{AES}_{S_\rho}(2) \)
3: \( R = \text{AES}_{S_\rho}(3) || \text{AES}_{S_\rho}(4) \)
4: \( \text{return } (S', R) \)

- \( S = (S_\rho, (S_i)_{i=0}^{31}), |S| = 33024 \)
- Involves a **Seed Extractor** a **Seeded Accumulator** and a **Scheduler**
- Requires a public random \( (X, X') \) of length 2048 bits
- Inputs shall be independent from \( X \)
- Leakage Security shall be studied: **SPOF**: \( S_\rho, |S_\rho| = 256 \)
# Model and constructions analysis

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Definition</th>
<th>Property</th>
<th>Attacker Capabilities</th>
<th>Construction</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BY03</td>
<td>1: $S \leftarrow \text{key}$</td>
<td>FWD</td>
<td>next-ror, get-state</td>
<td></td>
<td>AES (2)</td>
</tr>
<tr>
<td></td>
<td>2: $(S', R) \leftarrow \text{next}(S)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMOPST14</td>
<td>1: $S \leftarrow \text{key}$</td>
<td>LPR($f$)</td>
<td>next-ror, leaknext</td>
<td></td>
<td>AES (3)</td>
</tr>
<tr>
<td></td>
<td>2: $(S', R) \leftarrow \text{next}(S)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DHY02</td>
<td>1: $(K, S) \leftarrow \text{key}$</td>
<td>CIA</td>
<td>getinput, get-state, setinput</td>
<td></td>
<td>+ (3), × (2), AES (2)</td>
</tr>
<tr>
<td></td>
<td>2: $(S', R) \leftarrow \text{next}(S, K, I)$</td>
<td>CSA</td>
<td>getinput, get-state, set-state</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>KKA</td>
<td>getinput, get-key</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BST03</td>
<td>1: seed $\leftarrow \text{setup}$</td>
<td>RES($F$)</td>
<td>next-ror</td>
<td></td>
<td>× (1), [ ] (1)</td>
</tr>
<tr>
<td></td>
<td>2: $R \leftarrow \text{next}(\text{seed}, I)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BH05</td>
<td>1: $S' \leftarrow \text{refresh}(S, I)$</td>
<td>ROB($F$)</td>
<td>good-refresh, bad-refresh,</td>
<td></td>
<td>× (1), [ ] (1), ⊕ (1)</td>
</tr>
<tr>
<td></td>
<td>2: $(S', R) \leftarrow \text{next}(S)$</td>
<td></td>
<td>get-state, next-ror</td>
<td></td>
<td>AES (2)</td>
</tr>
<tr>
<td>DPRWV13</td>
<td>1: seed $\leftarrow \text{setup}$</td>
<td>ROB($\gamma^*$)</td>
<td>$D$-refresh, set-state, get-state</td>
<td></td>
<td>× (1), + (1)</td>
</tr>
<tr>
<td></td>
<td>2: $S' \leftarrow \text{refresh}(\text{seed}, S, I)$</td>
<td></td>
<td>next-ror</td>
<td></td>
<td>× (1), [ ] (1), AES (9)</td>
</tr>
<tr>
<td></td>
<td>3: $(S', R) \leftarrow \text{next}(\text{seed}, S)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSSW14</td>
<td>1: seed $\leftarrow \text{setup}$</td>
<td>NROB($\gamma^*, \beta$)</td>
<td>$D$-refresh, set-state, get-state</td>
<td></td>
<td>+ (1), × (2), ⊕ (1), [ ] (2), AES (11)</td>
</tr>
<tr>
<td></td>
<td>2: $S' \leftarrow \text{refresh}(\text{seed}, S, I)$</td>
<td></td>
<td>next-ror</td>
<td></td>
<td>AES (4)</td>
</tr>
<tr>
<td></td>
<td>3: $(S', R) \leftarrow \text{next}(\text{seed}, S)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

Contribution

- Revisited the notions of Extractors and Accumulators
- Unified the presentation of PRNG models
- Proposed AES based constructions
- Identified a potential vulnerability in NIST CTR_DRBG

Perspectives

- Independence requirement ?
- Leakage security of [DSSW14] construction ?
- Lightweight PRNG ?