Extension of Barack Halevi model and applications

Sylvain Ruhault (ENS, Oppida)

11/10/2012

Join work with Yevgeniy Dodis (NYU), David Pointcheval (ENS) and Damien Vergnaud (ENS)

Agenda

Need for randomness in concrete situations

Needs

- (Session, root, servers) keys generation
- Encryption : RSA paddings, El Gamal, CBC mode
- Signature : DSA
- Nonces in security protocols e.g. TLS, IPSEC

Tools for randomness generation

- Network devices
- Isolated servers
- Dedicated crytographic software or hardware
- Java applets, web browsers

Need for randomness in concrete situations

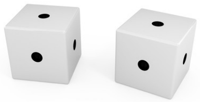
Implementation example: TLS protocol

- TLS protocol needs randomness:
  - Exchange, session, signature keys generation
  - Nonces, paddings, initialisation vectors generation
- Typical server implementation uses Apache mod_ssl module on a Linux server
- Typical client implementation uses browser or Java applet

Need for randomness in concrete situations

Recent vulnerabilities

- Implementation vulnerabilities
  - "Ron was wrong, Whit is right"
  - Openssl Debian implementation
- Attacks using bad PRNG
  - DSS private signature key recovery: when a LCG is used, 3 signatures can help signature forgery
  - RSA OAEP with e=3 is not one way when used with poor randomness

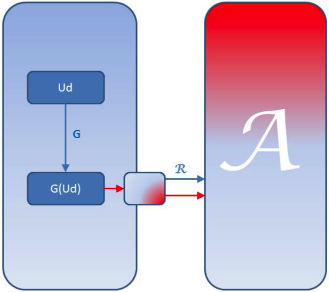


Definitions

Pseudorandom generator

A function $G : \{0,1\}^d \to \{0,1\}^m$ is a pseudorandom generator if

- $m \gg d$ (G expands)
- Output of a truly random seed is indistinguishable from random



$\exists \epsilon, \forall$ PPT $A, \forall n,$

$$|Pr[A(G(U_{d(n)})) = 1] - Pr[A(U_{m(n)}) = 1]| \leq \epsilon(n)$$

$$00100111010 \xrightarrow{G} 0010111010110100101101001001011010011010110$$

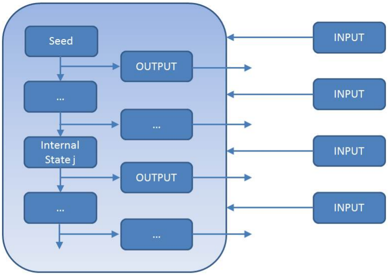
Definitions

Generator without input

- Seed S_0
- Successive outputs of G with a deterministic function
- Examples: LCG, DSA generator

Generator with input

- Additionnal data used to refresh the internal state of the generator
- Examples: DSA, Linux, Openssl, Java generators

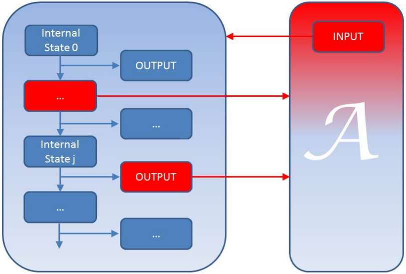


Security models

Associated security models

Attacker can interact with generator G with 3 interfaces:

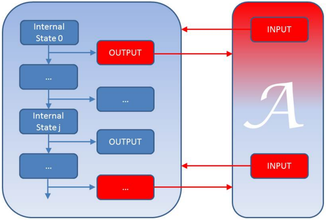
- Input control
- Internal state compromise
- Output request



Security models

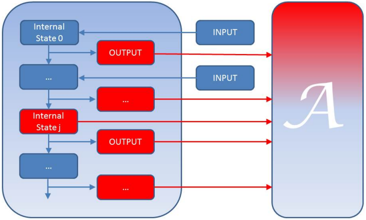
Resilience

- Potentially total control of the input data
- No access to internal state
- Output request



Backward and forward security

- Internal state compromise
- Forward security: past outputs requests
- Backward security: future outputs requests



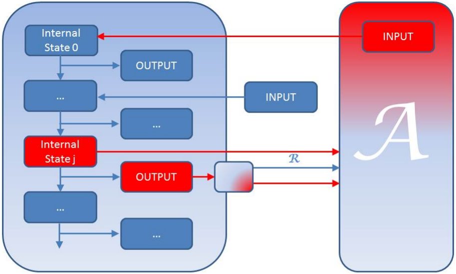
Security models

Associated security models

Generator is

- Resilient, or
- Backward secure, or
- Forward secure,

if A can't distinguish generator output from random output.



Relations between security properties

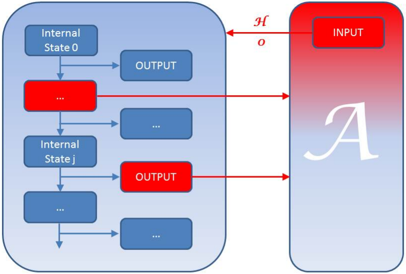
No implication between resilience, backward security and forward security

Security models

Barak-Halevi model

Attacker can interact with generator G with 4 interfaces:

- Input control:
  - no entropy input
  - high entropy input
- Internal state compromise
- Output request

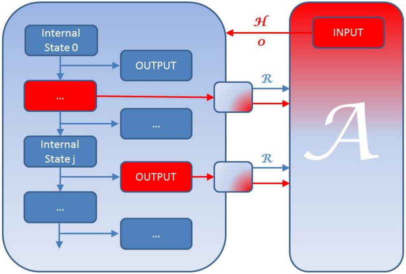


Security models

Barak-Halevi model

Generator is robust if, once G is refreshed with a high entropy input, A can't distinguish :

- state from random on state compromise
- generator output from random output on output request



Relations between security properties

Robustness implies resilience, backward security and forward security

Entropy definitions

High entropy input ?

Shannon Entropy: $H_1(X) = \sum_{x \in X} P[X = x] \times \log_2(\frac{1}{Pr[X=x]})$

- $X : \{0,1\}^{128} \to \{0,1\}^{128}$
- $Pr[X = 0] = 2^{-15}$
- $Pr[X = y, y \neq 0] = \frac{1-2^{-15}}{2^{128}-1}$

Then $H_1(X) = 127,997$

But ...

- A key K generated with this distribution. Then adversary A has probability 2^{-15} of guessing it by deriving it from $x = 0$
- If 2^{15} keys are generated with this distribution, then probability that one key is derived from $x = 0$ is $1 - e^{-1} \approx 0.63$

Entropy definitions

High entropy = High Min-Entropy

- Min-Entropy: $H_\infty(X) = \min_{x \in X} \left\{ \log_2\left(\frac{1}{Pr[X=x]}\right) \right\}$
- Computational Min-Entropy: $H_c(X) \geq k$,
  - $\exists Y, H_\infty(Y) = k$
  - $\exists \epsilon, \forall A, \forall n, Pr[A(X) = 1] - Pr[A(Y) = 1]| \leq \epsilon(n)$

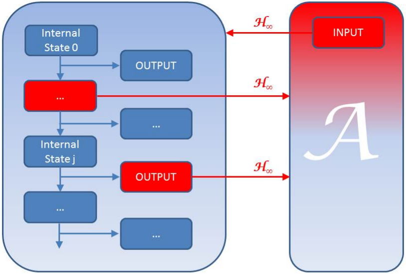
$H_\infty(X) = 15$

with distribution X

Analysis

Barak-Halevi model analysis

- Attacker should be able to interact with any Min-Entropy input.
- Min-entropy should be guaranteed after compromise



Model extension

Entropy preservation

A pseudorandom generator G ϵ-preserves $\{1, \infty, c\}$-entropy if:

- Entropy is preserved on state refresh
  - $H^*(S'|I) \geq H^*(S) - \epsilon$
  - $H^*(S'|S) \geq H^*(I) - \epsilon$
- Entropy is preserved on output request
  - $H^*(O) \geq H^*(S) - \epsilon$
  - $H^*(S'|O) \geq H^*(S) - \epsilon$

Refinement

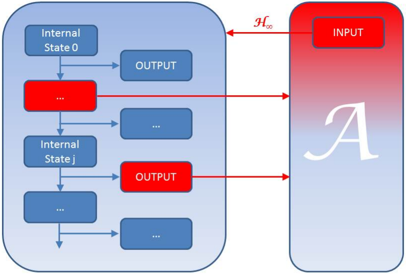
- Definition applicable for all entropy definitions, however not relevant for Shannon Entropy
- If all properties are requested, $H^* = H_c$

Model extension

Entropy preservation model

Attacker can interact with generator G with 3 interfaces:

- Input control: any entropy input
- Internal state compromise
- Output request

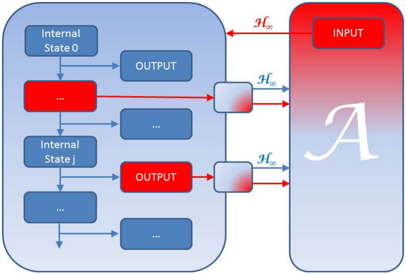


Model extension

Entropy preservation model

Generator preserves entropy if A can't distinguish generator output from output with given entropy:

- on state compromise
- on output request



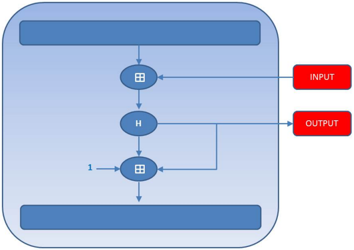
Theorem

H_c 0-preservation $\implies$ robustness

Application to DSA Generator analysis

Description

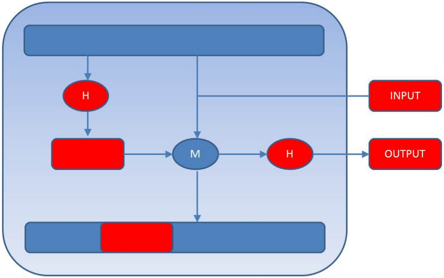
- Optional input
- Output generation:
  - $O = H((S + I) \mod 2^{160})$
  - $S' = (S + O + 1) \mod 2^{160}$

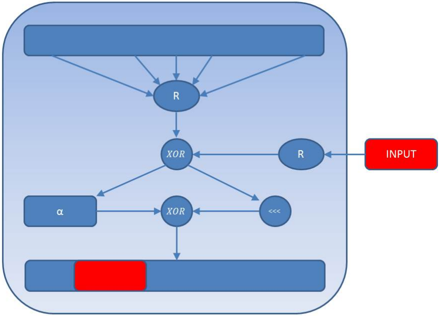


Theorem

- *If H is a random oracle $\Longrightarrow H_\infty$ 0-preservation*
- *If H is collision resistant $\Longrightarrow H_c$ 1-preservation, if $H_c(I) > 8$*

Application to Linux PRNG analysis





Theorem

- *If H is a random oracle $\Longrightarrow H_\infty$ 0-preservation*
- *If H is collision resistant $\Longrightarrow H_c$ 1-preservation, if $H_c(I) > O(1)$*

Conclusion

New security model for PRNG analysis and applications

- Extension of Barak-Halevi model
- Use of Min-Entropy
- Applications: security analysis of DSA and Linux Generators

Future work

- Security analysis of Openssl and Java Generators and others (virtual or embedded system)
- Supplementary security property: entropy accumulation

Thanks for your attention