# XDRBG: A Proposed Deterministic Random Bit Generator Based on Any XOF

**John Kelsey**[1,2], **Stefan Lucks**[3], **Stephan Müller**[4]

[1] NIST, Gaithersburg, USA, [2] COSIC, KU Leuven, Belgium
[3] Bauhaus-Universität Weimar, Germany
[4] atsec information security corp. Austin, USA

March 23, 2024

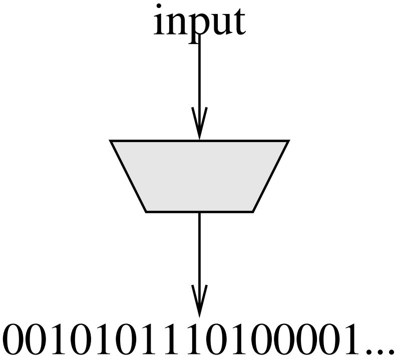
# Overview


What is a XOF? What is a DRBG?
Approach and Results
Refinements
Proposals and Performance
Conclusion and Outlook

# What is a XOF?

eXtended Output Function

hash function (random oracle) with an infinite number of output bits



# Details (XOF)

- hash function $H : \{0,1\}^* \to \{0,1\}^n$ for $n$ bits of output,
- XOF $H : \{0,1\}^* \to \{0,1\}^*$ can be written as family of functions

$$H(\ell, \cdot) : \{0,1\}^* \to \{0,1\}^\ell$$

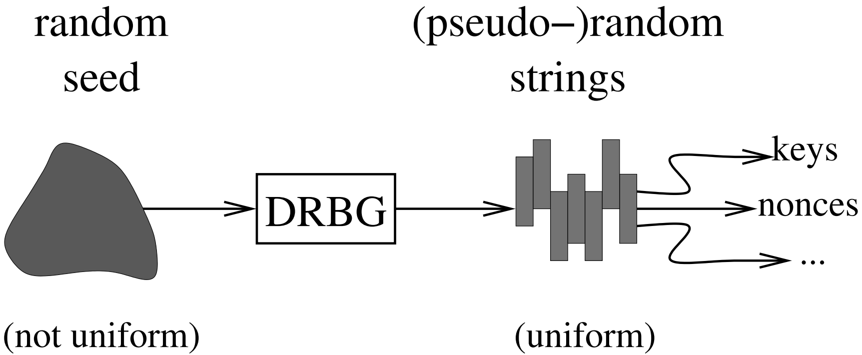
- if $\ell_1 \leq \ell_2$
  then for all $S \in \{0,1\}^*$:

$$H(\ell_1, S) \text{ is prefix of } H(\ell_2, S)$$

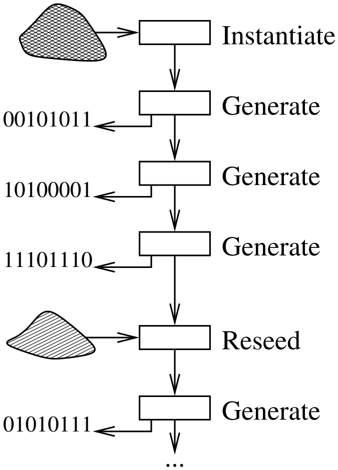
- if the XOF-*capacity* is $c$ bit and we model the XOF as a random oracle then we can claim up to $c/2$ bit of (classical) security

# What is a Deterministic Random Bit Generator (DRBG)?

pseudo-random strings from nonuniform random seeds (Zener diode, Geiger counter, ...)
in our case: known min-entropy for seed

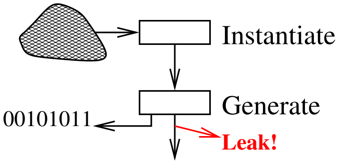


# Details (DRBG)



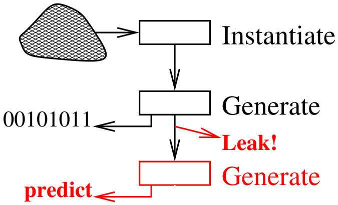

- fixed-size internal state
- three operations:
  1. **Instantiate**: seed $\to$ state
     min-entropy for seed: $H_{\text{init}} \leq |\text{state}|$
  2. **Generate**: state $\to$ (new state) $\times$ output
  3. **Reseed**: seed $\times$ state $\to$ (new state)
     min-entropy for seed: $H_{\text{rsd}} \leq |\text{state}|$
- output indistinguishable from random

# Details (DRBG)



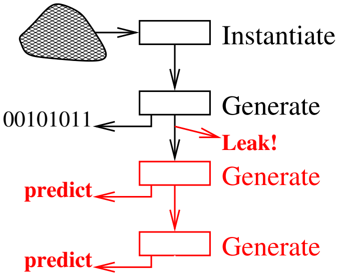

- fixed-size internal state
- three operations:
  1. **Instantiate**: seed $\rightarrow$ state
     min-entropy for seed: $H_{\text{init}} \leq |\text{state}|$
  2. **Generate**: state $\rightarrow$ (new state) $\times$ output
  3. **Reseed**: seed $\times$ state $\rightarrow$ (new state)
     min-entropy for seed: $H_{\text{rsd}} \leq |\text{state}|$
- output indistinguishable from random
- if a state is compromised
  - previous outputs are still pseudorandom
    (e.g., cryptographic keys not compromised)

# Details (DRBG)



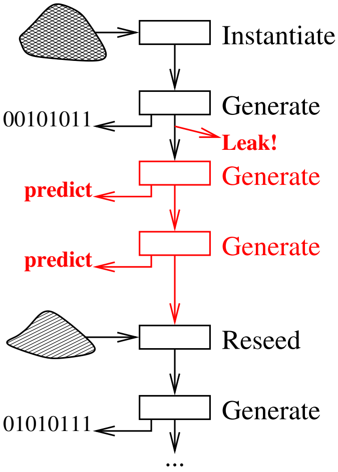

- fixed-size internal state
- three operations:
  1. **Instantiate**: seed $\rightarrow$ state
     min-entropy for seed: $H_{\text{init}} \leq |\text{state}|$
  2. **Generate**: state $\rightarrow$ (new state) $\times$ output
  3. **Reseed**: seed $\times$ state $\rightarrow$ (new state)
     min-entropy for seed: $H_{\text{rsd}} \leq |\text{state}|$
- output indistinguishable from random
- if a state is compromised
  - previous outputs are still pseudorandom
    (e.g., cryptographic keys not compromised)
  - the next ouputs are predictable

# Details (DRBG)



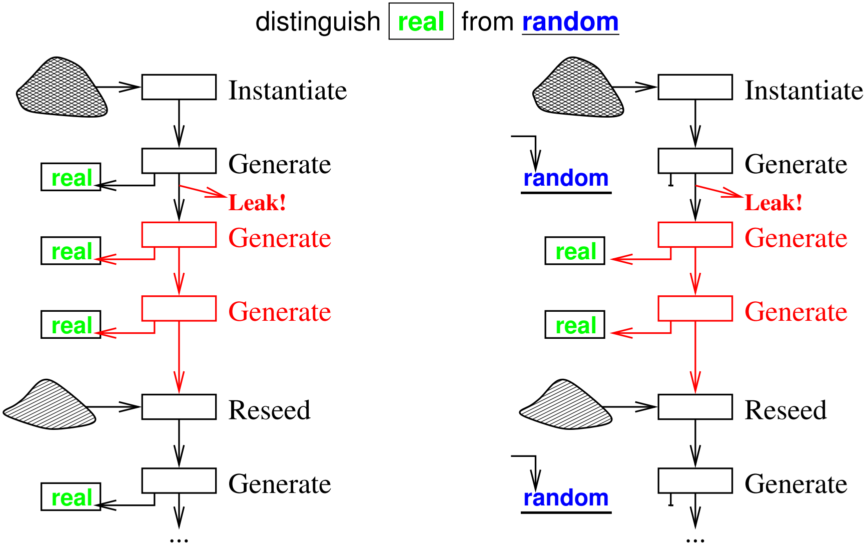

- fixed-size internal state
- three operations:
  1. **Instantiate**: seed $\rightarrow$ state
     min-entropy for seed: $H_{\text{init}} \leq |\text{state}|$
  2. **Generate**: state $\rightarrow$ (new state) $\times$ output
  3. **Reseed**: seed $\times$ state $\rightarrow$ (new state)
     min-entropy for seed: $H_{\text{rsd}} \leq |\text{state}|$
- output indistinguishable from random
- if a state is compromised
  - previous outputs are still pseudorandom
    (e.g., cryptographic keys not compromised)
  - the next ouputs are predictable

# Details (DRBG)




- fixed-size internal state
- three operations:
  1. **Instantiate**: seed $\to$ state
     min-entropy for seed: $H_{\text{init}} \leq |\text{state}|$
  2. **Generate**: state $\to$ (new state) $\times$ output
  3. **Reseed**: seed $\times$ state $\to$ (new state)
     min-entropy for seed: $H_{\text{rsd}} \leq |\text{state}|$
- output indistinguishable from random
- if a state is compromised
  - previous outputs are still pseudorandom (e.g., cryptographic keys not compromised)
  - the next ouputs are predictable
  - but the DRBG recovers after a Reseed (or an Instantiate)

# DRBG Attack Model



# Approach

- *R* *requests* from adversary
  $\rightarrow$ one request (Instantiate, Generate, Reseed) $\rightarrow$ one XOF call
- *Q* direct XOF *queries* made by adversary
- attacker benefits from certain *bad events* (no bad event, no win)

# Bad Events and their Approximate Probabilities

- two request with same input

$$\frac{R^2}{2^{H_{\text{init}}}}$$

# Bad Events and their Approximate Probabilities

- two request with same input

$$\frac{R^2}{2^{H_{\text{init}}}}$$

- query matches Instantiate or any uncompromised request

$$\frac{Q \cdot R}{2^{H_{\text{init}}}}$$

# Bad Events and their Approximate Probabilities

- two request with same input

$$\frac{R^2}{2^{H_{\text{init}}}}$$

- query matches Instantiate or any uncompromised request

$$\frac{Q \cdot R}{2^{H_{\text{init}}}}$$

- query matches (compromised) Reseed request

$$\frac{Q}{2^{H_{\text{rsd}}}}$$

# Dominating Terms for Security Bound

## . . . and recommendation for required entropy

$$\text{bound} \approx \frac{R^2}{2^{H_{\text{init}}}} + \frac{Q \cdot R}{2^{H_{\text{init}}}} + \frac{Q}{2^{H_{\text{rsd}}}}$$

$$\boxed{H_{\text{rsd}} \approx H_{\text{init}} - \log_2(R)}$$

# Refinements

- *multiple devices* running in parallel
- some devices may be compromised, when others aren't

# Refinements

- *multiple devices* running in parallel
- some devices may be compromised, when others aren't

- *additional input* $\alpha$ for each operation (aka request)
- thee scenarios
  1. *fixed* $\alpha$ (e.g., empty string)
  2. $\alpha$ as a *nonce* (never used twice)
  3. *personalization:*
     - each device gets it's own $\alpha$
     - the same $\alpha$ for all requests to the same device
     - at most $R_{\text{Device}}$ requests to any single device

# Refined Results and Recommendations

see paper for exact bounds

1. fixed (as seen before)

$$\boxed{H_{\mathrm{rsd}} \approx H_{\mathrm{init}} - \log_2(R)}$$

$$\frac{Q \cdot R}{2^{H_{\mathrm{init}}}} + \frac{Q}{2^{H_{\mathrm{rsd}}}} + \frac{R^2}{2^{H_{\mathrm{init}}}}$$

2. nonce

$$\boxed{H_{\mathrm{init}} = H_{\mathrm{rsd}}}$$

$$\frac{Q}{2^{H_{\mathrm{init}}}} + \frac{Q}{2^{H_{\mathrm{rsd}}}} + \frac{R^2}{2^{|\mathrm{state}|}}$$

3. personalization

$$\boxed{H_{\mathrm{rsd}} \approx H_{\mathrm{init}} - \log_2(R_{\mathrm{Device}})}$$

$$\frac{Q \cdot R_{\mathrm{Device}}}{2^{H_{\mathrm{init}}}} + \frac{Q}{2^{H_{\mathrm{rsd}}}} + \frac{R \cdot R_{\mathrm{Device}}}{2^{H_{\mathrm{init}}}}$$

# Our Proposals

## Based on SHAKE ($\rightarrow$ SHA-3 standard) and Ascon

| | capacity | $H_{\text{init}}$ | $H_{\text{rsd}}$ | $\log_2(R)$ | $\log_2(R_{\text{Device}})$ | promised security level: classical | promised security level: quantum (Grover) | promised security level: category |
|---|---|---|---|---|---|---|---|---|
| SHAKE: | | | | | | | | |
| XDRBG-128 | 256 | 192 | 128 | 128 | 56 | 128 | 64 | 1 |
| XDRBB-192 | 512 | 240 | 240 | 128 | 56 | 192 | 96 | 3 |
| XDRBG-256 | 512 | 384 | 256 | 128 | 128 | 256 | 128 | 5 |
| Ascon: | | | | | | | | |
| XDRBG-L-128 | 256 | 192 | 128 | 128 | 56 | 128 | 64 | 1 |
| XDRBG-L-170 | 308 | 240 | 240 | 128 | 64 | 170 | 85 | 2 |

$H_{\text{init}}$, $H_{\text{rsd}}$: required min-entropy for Instantiate/Reseed
$R$, $R_{\text{Device}}$: number of requests in total / for each device
category: NIST post-quantum security level (1: min, 5: max)

# Performance (e.g. for XDRBG-256)

fastest one red second fastest blue

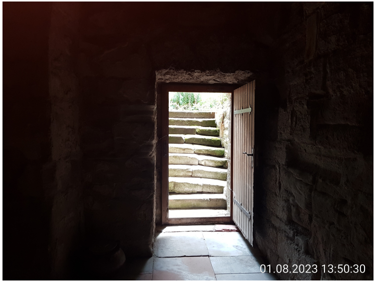
| | XDRBG-256 SHAKE256 Vec. Instr. | XDRBG-256 SHAKE256 | HashDRBG SHA256 | HMACDRBG SHA256 |
|---|---|---|---|---|
| AMD Ryzen 5950X | 4.62 | 5.06 | 6.63 | 27.33 |
| Intel 11th Gen i7-1195G7 | 2.64 | 5.28 | 5.81 | 23.11 |
| Intel 12th Gen i7-1280P | 3.96 | 4.15 | 4.68 | 19.97 |
| Apple M2 | 2.18 | 2.89 | 4.88 | 20.54 |
| ARM Cortex-A76 r4p1 | 6.17 | 6.52 | 9.59 | 41.15 |
| ARM Cortex-A72 r0p3 | 12.26 | 12.33 | 18.15 | 78.20 |
| ARM Cortex-A8 r2p5 | 62.68 | 185.45 | 186.71 | 784.66 |
| ARM Cortex-A7 r0p5 | 81.81 | 249.85 | 242.79 | 1015.23 |
| Si Five (RISC-V) | 104.73 | 104.80 | 72.11 | 309.03 |

# Conclusion and Outlook

- the XDRBG is a Ditigal Random Bit Generator
- the approach is very generic: the XDRBG can be based on any XOF,
  $\rightarrow$ not limited to SHAKE and Ascon
- the XDRBG has been proven secure in the random oracle model
  $\rightarrow$ personalization (unique $\alpha$ for each device) for improved bounds

# Conclusion and Outlook

- the XDRBG is a Ditigal Random Bit Generator
- the approach is very generic: the XDRBG can be based on any XOF,
  $\rightarrow$ not limited to SHAKE and Ascon
- the XDRBG has been proven secure in the random oracle model
  $\rightarrow$ personalization (unique $\alpha$ for each device) for improved bounds




- we expect the SHAKE-based instantiation of the XDRBG to match future requirements for standards currently under revision, such as
  - SP 800-90 (NIST, USA) and
  - AIS 20/31 (BSI, Germany)