Controlled Randomness – A Defense against Backdoors in Cryptographic Devices

Lucjan Hanzlik, Kamil Kluczniak, Mirosław Kutyłowski

Wrocław University of Science and Technology, Poland

MYCRYPT -Paradigm Shifting Cryptography 2016, Kuala Lumpur

Role of randomness

Randomness in cryptographic protocols

- most signature schemes, even deterministic ones (key generation, padding, . . .)
- challenge-response protocols
- DH key agreement
- . . .

removing randomness from crypto seems to be as difficult as building post-quantum systems (or even more difficult)

Catacrypt and randomness

What if randomness source not ideal?

- while designing a scheme one concerns the randomness a ideal one
  do ideal sources exist in reality?
- what happens if the randomness is not ideal?

Catacrypt

advances in attack technology leading to severe failure of cryptography

- is catacrypt a potential future, or . . .
- . . . it has already happened?

Randomness and secure devices

current approach

- if possible implement in black-box hardware
- tamper-evident or tamper-proof devices
- randomness tests/ certification / inspection by authorities to ensure proper design

Certification/audit

problems with certification /audit

- requires insight into industrial secrets
- tedious and expensive
- not verifiable by an end-user
- the manufacturer, the certification body and supervisory authorities may collude against a user

From the point of view of an end-user accepting certification result is **based on trust and not on evidence**

local verifiability

the user should be able to check whether device security level is relevant for a concrete application

Threats

Hardware Trojans

- inspection of the chip under microscope, layer by layer, does not reveal any inconsistency with the implementation documentation
- . . . yet the randomness in some sense predictable by the attacker

Kleptographic code

- malicious cryptography
- deviations from the protocol but undetectable for the user
- e.g.: subsequent choices of random numbers entangled in a cryptographic way – an adversary holding a secret key may exploit it

RNG versus PRNG

True Random Number Generator (RNG)

- based on physical effects
- hard to build a source with uniform distribution
- even harder to test:
  - regular randomness tests detect major failures
  - useless against malicious constructions

recommendations

- not to be used alone
- use together with PRNG as a source of extra randomness

PRNG

Pseudorandom Number Generator (PRNG)

- verifiable – set the seed and check the output
- but how to initialize the seed?

Options for setting the seed

option 1: the manufacturer **installs** the seed,
no protection against malicious manufacturer

option 2: the user creates the seed by starting a procedure executed **internally** by the PRNG
the process might be a fake – the same concerns as for option 1

option 3: the **user** uploads the seed to the PRNG
the user is also a potential adversary and may try to get access to the secrets from the device

Options for setting the seed

option 4: the user uploads a **part** of the seed while the second part of the seed is installed by the manufacturer,
how to check that each part is used properly?

option 5: the user and/or the manufacturer uploads the seed, however, during its operation the PRNG modifies its state according to some number of **entropy** bits.
the changes may gradually convert into a seed predictable by the adversary

PRNG
security situation

Current situation

no guarantees that the PRNG is secure *by-design*

an adversary may know/guess/predict its internal state

Our goal

find effective countermeasures
but avoid rebuilding cryptography from scratch – no time, no resources available

Scenarios to use random numbers

option 1 choose random r and make it available to other participants

explicitly or implicitly addressed in the literature

option 2 choose random k, compute $r := g^k$ and present r the other party in the protocol

our focus

option 3 choose random r and use it deterministically but not present it to other parties

a challenging problem, e.g. RSA key generation process

Controlled randomness

Idea

- the output of PRNG not used directly but subject of deterministic modification based on blinding key set by the user
- user gets control data from the device
- control data not forwarded to other protocol participants

Device setup

- a PRNG P with a seed y installed by the manufacturer
- a *blinding factor* $U = g^u$ installed on the device by its owner
- u never exposed to the device

Generating r

- k_0 is taken as the output of P,
- $k_1 := \mathrm{Hash}(U^{k_0}, i)$,
  - Hash is a cryptographic hash function with results in the range $[0, q-1]$
  - i is a counter
- $r' := g^{k_0}$,
- $r := (r')^{k_1}$

Verification of r

On input r and control parameters (r', i), the user performs the following steps:

- $\lambda := \mathrm{Hash}((r')^u, i)$
- if $r \neq (r')^\lambda$, then consider the device as *faulty* or *malicious*.

note that $(r')^u = (g^{k_0})^u = (g^u)^{k_0} = U^{k_0}$
(kleptographic trick by Young and Yung)

Schnorr Signature

setup: private key x and public key $y = g^x$

signature creation:

$$
\begin{aligned}
k &:= \mathrm{prng}(), \quad r := g^k \\
e &:= \mathrm{Hash}(m||g^r) \\
s &:= (k - x \cdot e) \bmod q
\end{aligned}
$$

Schnorr Signature with controlled randomness

$$
\begin{aligned}
k_0 &:= \mathrm{prng}() \\
r' &:= g^{k_0} \bmod p \\
k_1 &:= \mathrm{Hash}(U^{k_0}, i) \\
k &:= k_0 \cdot k_1 \\
r &:= g^{k} \bmod p \\
e &:= \mathrm{Hash}(m || g^{r}) \\
s &:= (k - x \cdot e) \bmod (p-1)
\end{aligned}
$$

- (s, e) is the signature,
- the control data are (r', i)

Example: Diffie-Hellmann with controlled randomness

the device A of Alice executes the following operations:

1. choose k at random (take the output from the PRNG),
2. $preY_A := g^k$,
3. $k' := \mathrm{Hash}(U^k, i)$,
4. $Y_A := (preY_A)^{k'}$,
5. $y_A := k \cdot k' \bmod q$, where q is the order of the group used

Y_A is presented by the device A together with $preY_A$ and i

Example: PACE with CR

Card	Controller	Reader
holds: password π counter i	password π	holds: password π entered by the Card owner
	Card Setup with the Controller	
	choose $u, v, w, d < q$ at random $U := g^u$, $V := g^v$, $W := g^w$, $D := g^d$	
	$\leftarrow$ U, V, W, D	
install U, V, W, D	retain u, v, w, d for control purposes	

Example: PACE with CR

Card		Controller		Reader
Authentication Session				
$K_\pi := \mathrm{Hash}(0\|\|\pi)$		$K_\pi := \mathrm{Hash}(0\|\|\pi)$		$K_\pi := \mathrm{Hash}(0\|\|\pi)$
$i := i + 1$				
choose s at random				
$z := \mathrm{Enc}(K_\pi, s)$				
$\delta := \mathrm{prng}(),\ \Delta := g^\delta$				
$z := \mathrm{Hash}(D^\delta, i)$				
$s := \mathrm{Dec}(K_\pi, z)$				
	$\xrightarrow{}$ $\mathcal{G}, z, \Delta$		$\xrightarrow{}$ $\mathcal{G}, z$	abort if $\mathcal{G}$ incorrect
		control test:		$s := \mathrm{Dec}(K_\pi, z)$
		$z \stackrel{?}{=} \mathrm{Hash}(\Delta^d, i)$		
		....		

Example: PACE with CR

Card	Controller	Reader
	Authentication Session	
choose $y_A \in \mathbb{Z}_q$ at random		choose $y_B \in \mathbb{Z}_q$ at random
$k_0 := \mathrm{prng}(),\ K_0 := g^{k_0}$		
$k_1 := \mathrm{Hash}(U^{k_0}, i, 1)$		
$y_A := k_0 \cdot k_1$		
$Y_A := g^{y_A}$		$Y_B := g^{y_B}$
	$\xleftarrow{Y_B}$ $\xrightarrow{Y_A,\ K_0, i}$ (Card ↔ Controller); $\xleftarrow{Y_B}$ $\xrightarrow{Y_A}$ (Controller ↔ Reader)	
	control test: $Y_A \stackrel{?}{=} K_0^{\mathrm{Hash}(K_0^u, i, 1)}$	
$h := Y_B^{y_A}$ $\hat{g} := h \cdot g^s$		$h := Y_A^{y_B}$ $\hat{g} := h \cdot g^s$
	....	

$v_0 := \mathrm{prng}(),\ V_0 := g^{v_0}$

choose $y'_B \in \mathbb{Z}_q$ at random

$w_0 := \mathrm{prng}(),\ W_0 := g^{w_0}$

$\kappa := \mathrm{Hash}(V^{v_0}, i, 1)$

$t_0 := \mathrm{prng}(),\ T_0 := \hat{g}^{t_0}$

$C := \mathrm{Enc}_\kappa(T_0)$

$t_1 := \mathrm{Hash}(W^{w_0}, C, i, 2)$

$y'_A := t_0 \cdot t_1$

$Y'_A := \hat{g}^{y'_A}$

$Y'_B := \hat{g}^{y'_B}$

$\overleftarrow{Y'_B}$ $\overleftarrow{Y'_B}$

check $Y'_B \neq Y_B$

$\overrightarrow{V_0, W_0, C, Y'_A}$ $\overrightarrow{Y'_A}$

check $Y'_A \neq Y_A$

control test:

$\kappa := \mathrm{Hash}(V_0^v, i, 1)$

$T_0 := \mathrm{Dec}_\kappa(C)$

$t_1 := \mathrm{Hash}(W_0^w, C, i, 2)$

$Y'_A \stackrel{?}{=} T_0^{t_1}$

$K := {Y'_B}^{y'_A}$ $K := {Y'_A}^{y'_B}$

.... *FINAL STAGE* ....

Manufacturer Mallet

Assumptions

- Mallet knows output of PRNG
- he does not know the blinding key

Theorem

Mallet **cannot distinguish** between Schnorr signatures created by a device implementing CR from the Schnorr signatures created with the same signing key by a device with the standard implementation (no CR).

In the first case Mallet is given the output of the PRNG, in the second case Mallet is given a random output.

Malicious user

Threat

potentially the user may steal own key **as he gets more output** from the signing device.

Theorem

If there is a user that holds a device with CR **and then can create a valid signature without the device**,
then
the same holds for the regular Schnorr signatures.

Malicious device

Leaking key-bits in the regular case

- random components might be correlated via kleptographic techniques
- few bits leaked with each signature if the device has time to make a few trials

Proposition

Assuming KEA1 this is the only way to cheat.

Final remarks

- a user gets a **real opportunity** to check his devices
- it is **relatively simple** to make the changes in simple protocols
- for protocols where **the generator is changed** in a cryptographic way (like for PACE) the situation becomes complicated (protocol changes, proofs)

Thanks for your attention!

Contact data

1. Miroslaw.Kutylowski@pwr.edu.pl
2. http://kutylowski.im.pwr.edu.pl
3. http://cs.pwr.edu.pl