Backdoors in PRGs and PRNGs

Kenny Paterson

Information Security Group

@kennyog; www.isg.rhul.ac.uk/~kp

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Overview of this lecture

ROYAL HOLLOWAY UNIVERSITY OF LONDON

- Motivation for considering backdoors
- Backdoors in PRGs
- Backdoors in PRNGs (PRGs with entropy inputs)

Motivation

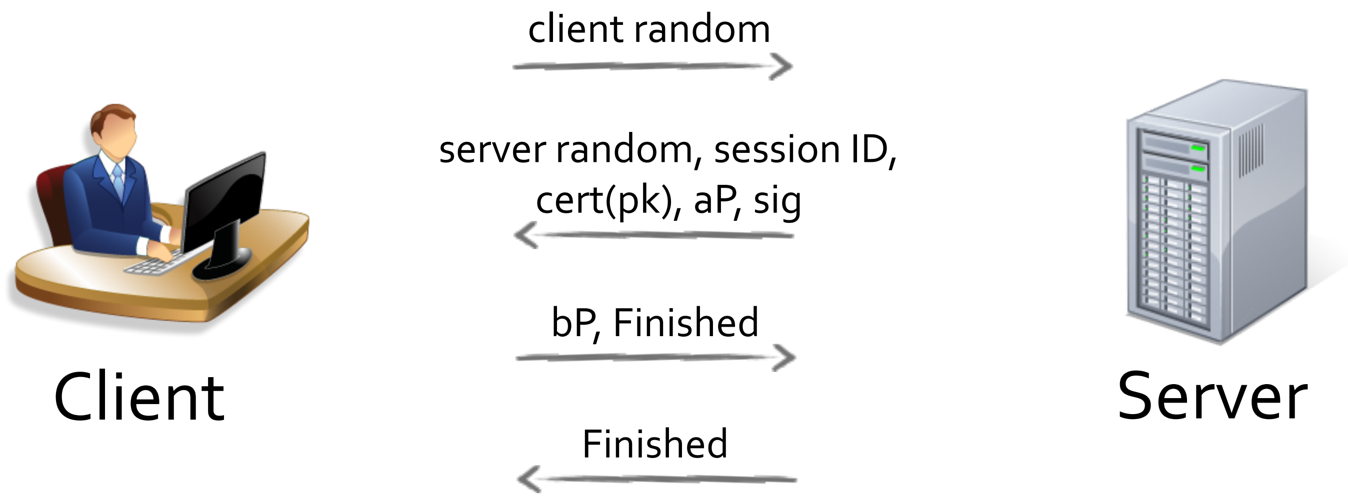
The Snowden revelations

- In 2013, Snowden revealed the extent of the NSA mass surveillance programs
- New threat model:
  - Backdoors, subversion, ...

- Led to increased suspicion of the Dual_EC pseudorandom generator
  - Standardized by several standardization bodies: NIST, ISO, ANSI, ...
  - Simple generator based on two (specific and fixed) elliptic curve points, P and Q.
  - Biased and slow, so no real incentive to use it.
  - But knowledge of the discrete log of P wrt. Q allows state recovery from generator outputs (Shumov-Ferguson 2007), so good target for backdooring.

Basis for an attack against TLS?

TLS ECDHE handshake (simplified):



MS = PRF(x(abP), "master secret", client random, server random)

Checkoway et al. "On the Practical Exploitability of Dual EC in TLS Implementations", USENIX'14

The Juniper incident

Juniper Networks is a major vendor of network security devices.

ScreenOS is the Operating System in Juniper's Netscreen VPN product family.

2008: Juniper adopt Dual_EC in ScreenOS.

10/2013: Juniper publish a knowledge base article explaining that ScreenOS uses Dual EC, but "in a way that should not be vulnerable to the possible issue that has been brought to light".

- Custom Q instead of NIST-standardised (and NSA-generated) Q.
- Dual_EC output post-processed by ANSI X9.31 generator.

12/2015: Juniper makes vulnerability announcement:

"VPN Decryption (CVE-2015-7756) may allow a knowledgeable attacker who can monitor VPN traffic to decrypt that traffic. [...] This issue affects ScreenOS 6.2.0r15 through 6.2.0r18 and 6.3.0r12 through 6.3.0r20. No other Juniper products or versions of ScreenOS are affected by this issue. There is no way to detect that this vulnerability was exploited".

The Juniper incident

2015/2016: Reverse engineering effort by Checkoway et al. discovers:

- Subtle scoping bug in code means that Dual_EC output is directly exposed as ScreenOS PRNG output (instead of being post-processed).
- Increased nonce size of 32 bytes in Juniper IKE implementation is ideal for recovering Dual_EC state.
- Even though nonce follows DH value in IKE protocol, nonce value is generated *before* DH value and stored in a queue.
- Hence, someone who knows $dlog_P(Q)$ can recover (EC)DH private value using Dual_EC backdoor, and thence all encryption keys, from observing a single IKE run.
- CVE-2015-7756 actually refers to a change in the Q value: it appears that Juniper's custom Q value was replaced in 2012, along with test vectors, by persons unknown.
- So Juniper (and possibly others) could passively break customers' IPsec traffic, but then lost the capability to persons unknown.

Details in: Checkoway et al., A Systematic Analysis of the Juniper Dual EC Incident, ACM-CCS 2016.

Backdoors in PRGs

Backdoors in PRGs

Main research question:

To what extent can provably secure pseudorandom generators be backdoored?

Two recent research papers addressing this:

- Dodis-Ganesh-Golovnev-Juels-Ristenpart (Eurocrypt 2015)
- Degabriele-Paterson-Schuldt-Woodage (Crypto 2016)

Pseudorandom Generators (PRGs)

Pseudorandom generator

Given a short random seed as input, a PRG outputs an arbitrary long string of pseudorandom bits

$1^\lambda \longrightarrow \boxed{\text{setup}} \longrightarrow (pp, bk)$

$pp \longrightarrow \boxed{\text{init}} \longrightarrow st$

$st \longrightarrow \boxed{\text{next}} \longrightarrow r, st'$

Forward Security for PRGs

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Game FWD(, q)

$1^\lambda \longrightarrow$ setup $\longrightarrow$ (pp,bk)

pp $\longrightarrow$ init $\longrightarrow$ st_0

$st_0 \longrightarrow [\,$next$\,]^q \longrightarrow (st_q, r_1^0, \ldots, r_q^0)$

$\longrightarrow (r_1^1, \ldots, r_q^1)$

$\{0,1\} \longrightarrow b$

$(\, pp, r_1^b, \ldots, r_q^b, st_q \,) \longrightarrow$ $\longrightarrow b'$

return(b = b')

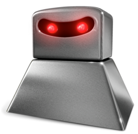
Advantage

Adv(, q) = 2| Pr[FWD ⇒ 1] - 1/2 |

(q, ε)-FWD Security

For all : Adv(, q) ≤ ε

Backdoored PRGs

Big Brother: 

Backdooring Game

Let type-BPRG( ) be game capturing a specific backdooring goal, and let Adv( ) denote the corresponding advantage.

(q, δ, [type, ε])-FWD-secure BPRG

A tuple of algorithms PRG′ = (setup, init, next,) is a (q, δ, [type, ε])-FWD-secure BPRG if:

- PRG = (setup, init, next) is a (q,δ)-FWD-secure PRG
- Adv( ) $\geq \varepsilon$

Dodis-Ganesh-Golovnev-Juels-Ristenpart (2015)

- Consideration of various different backdooring goals.
  - Distinguishing output from random: type = DIST
  - Prediction of past/future outputs given current output (random seek): type = RSEEK
  - Prediction of current state: type = NEXT
  - (In practice, BB would like to recover initial state, not addressed by Dodis et al.)
- Equivalence of DIST-backdoored PRGs and single-bit public key encryption with pseudorandom ciphertexts.
  - So backdoored PRGs are really public key primitives.
  - cf. use of ECDLP to build Dual_EC.
  - Means that constructions will "look suspicious".

DIST-BPRG game

Game DIST-BPRG(,q)

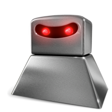
$1^\lambda \longrightarrow$ setup $\longrightarrow$ (pp,bk)

pp $\longrightarrow$ init $\longrightarrow$ st_0

$st_0 \longrightarrow [\text{next}]^q \longrightarrow (st_q, r_1^0, \ldots, r_q^0)$

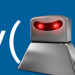
 $\longrightarrow (r_1^1, \ldots, r_q^1)$

$\{0,1\} \longrightarrow b$

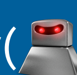
$(\mathbf{bk}, r_1^b, \ldots, r_q^b) \longrightarrow$  $\longrightarrow b'$

return (b = b′)

Advantage

Adv(, q) = 2| Pr[FWD ⇒ 1] - 1/2 |

(q, δ, [DIST, ε])-FWD-secure BPRG:

- PRG = (setup, init, next) is (q,δ)-FWD-secure.
- Adv(, q) ≥ ε

Construction of bit encryption using a backdoored PRG from [DGGJR15]

(q, δ, [DIST, ε])-secure BPRG

$1^\lambda \longrightarrow$ setup $\longrightarrow$ (pp,bk)

pp $\longrightarrow$ init $\longrightarrow$ st

st $\longrightarrow$ next $\longrightarrow$ r, st'

PKE

<u>KGen(1^λ):</u>

$1^\lambda \longrightarrow$ setup $\longrightarrow$ (pp,bk)

return(PK = pp, SK = bk)

<u>Enc(PK,b):</u>

PK $\longrightarrow$ init $\longrightarrow$ st_0

$st_0 \longrightarrow [\text{next}]^q \longrightarrow (st_q, r_1^0, \ldots, r_q^0)$

$\longrightarrow (r_1^1, \ldots, r_q^1)$

return$(r_1^b, \ldots, r_q^b)$

<u>Dec(SK,c):</u>

(SK,c) $\longrightarrow$ $\longrightarrow$ b'

return(b')

<u>Theorem:</u>
The construction produces a single-bit PKE scheme that is ε-correct and (q,δ)-IND-$CPA secure.

Further results in [DGGJR15]

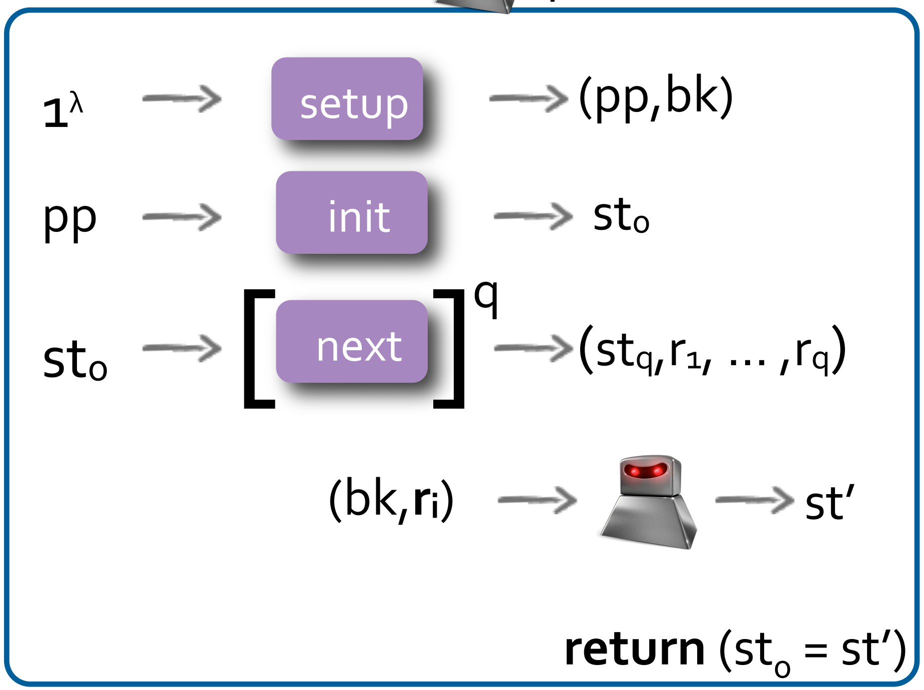
- Various constructions for backdoored PRGs for the different goals, DIST, RSEEK, NEXT.
- Careful study of "immunisation" of backdoored PRGs to remove backdoors.
- Highly relevant in light of the Juniper incident!

Open Problems:

- Can a BPRG be simultaneously forward secure and allow recovery of past outputs via backdooring?
- Can we achieve stronger backdooring notions for PRGs, like recovery of initial state?

FIRST-BPRG game from [DPSW16]

Game FIRST-BPRG(,q,i)



Advantage

Adv(,q ,i) = Pr[FIRST-BPRG ⇒ 1]

(q, δ, [FIRST, ε])-FWD-secure BPRG:

- PRG = (setup, init, next) is (q,δ)-FWD-secure.
- Adv(,q,i) ≥ ε for every i.

FIRST is a powerful backdooring notion: recovery of initial state st_0 from any output r_i allows reconstruction of **all past and future** outputs!

Building a FIRST-BPRG [DPSW16]

- A forward secure PRG = (setup', init', next')
- An IND\$-CPA secure reverse-rerandomizable encryption scheme PKE = (keygen, enc, rerand, rev-rerand, dec)

IND\$-CPA

Ciphertexts are indistinguishable from random strings

Rerandomizable

For all pk, m, r' :

$\{ \text{enc}(\text{pk}, m; r) \mid r \leftarrow R \} \approx \{ \text{rerand}(\text{enc}(\text{pk}, m; r'), r) \mid r \leftarrow R \}$

Reverse-rerandomizable

For all pk, m, r, r' :

$\text{enc}(\text{pk}, m; r) = \text{rev-rerand}(\text{rand}(\text{enc}(\text{pk}, m; r), r'), r')$

A FIRST-BPRG construction [DPSW16]

setup

$(pk, sk) \leftarrow$ keygen

$(pp', \perp) \leftarrow$ setup'

pp

bk

ret

init

$st_0 \leftarrow init'(pp')$

PRG = (setup, init, next) is a (q, δ, (FIRST, 1))-FWD-secure BPRG.
This follows from:

- Forward security of PRG′ = (setup′, init′, next′)
- IND$-CPA security and rerandomization security of PKE = (keygen, enc, rerand, rev-rerand, dec)
- Ability to recover r values and reverse the rerandomizations

next

st' | c'

c

$(r, st') \leftarrow next'(st)$

an:

o;

the

r values;

- Reverse the rerandomizations of c to obtain c_0.
- (Run the PRG forward to compute all outputs.)

Backdoors in PRNGs

PRNGs (sometimes: PRNGs with input)

ROYAL HOLLOWAY UNIVERSITY OF LONDON

PRNG

A PRG that allows state updates with inputs from an entropy source

$1^\lambda \rightarrow$ setup $\rightarrow$ (pp, bk)

pp $\rightarrow$ init $\rightarrow$ st

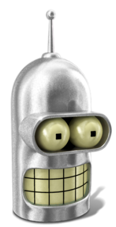
st $\rightarrow$ next $\rightarrow$ r, st'

(pp, st, I) $\rightarrow$ refresh $\rightarrow$ st'

Input from entropy source

Modeling entropy inputs: The distribution sampler [DPRVW13]

State: σ



$\sigma \longrightarrow$ Distribution sampler $\longrightarrow (\ \sigma',\ I,\ \gamma,\ z\ )$

Input to PRNG

Side information regarding input

Updated state

Entropy estimate for input

Entropy requirement:

$$H_\infty(\, I_i \mid I_1, \ldots, I_{i-1}, I_{i+1}, \ldots I_q, z_1, \ldots, z_q, \gamma_1, \ldots, \gamma_q\,) \geq \gamma_i$$

Robustness for PRNGs

Game ROB(, , γ^*)

$1^\lambda \rightarrow$ setup $\rightarrow$ (pp, bk)

pp $\rightarrow$ init $\rightarrow$ st

$\{0,1\} \rightarrow b$

$\varnothing \rightarrow \sigma$

$\infty \rightarrow c$

pp $\rightarrow$ (GET, SET, REF, ROR) $\rightarrow$ b′

return (b = b′)

REF( )

$\sigma \rightarrow$ $\rightarrow (\sigma, I, \gamma, z)$

(pp, st, I) $\rightarrow$ refresh $\rightarrow$ st

$c + \gamma \rightarrow c$

return(γ, z)

GET( )

$0 \rightarrow c$

return(st)

SET(st')

$0 \rightarrow c$

st′ $\rightarrow$ st

ROR( )

(pp,st) $\rightarrow$ next $\rightarrow$ r_0, st

$\rightarrow r_1$

if $c < \gamma^*$ then

$0 \rightarrow c$

return(r_0)

else

return(r_b)

Advantage

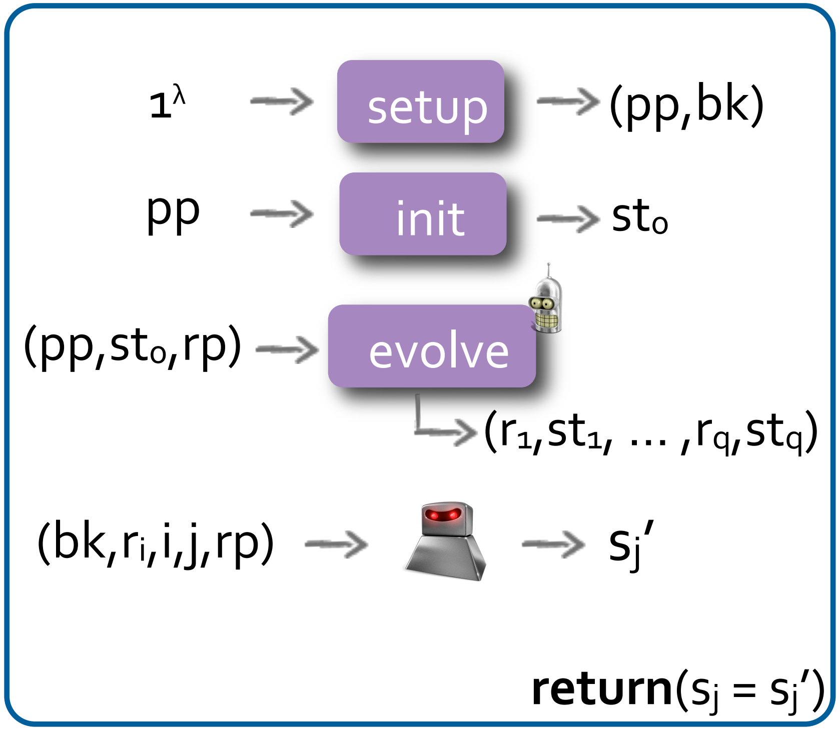
Adv(, , γ^*) :=

2| Pr[ROB(, , γ^*) $\Rightarrow$ 1] - 1/2 |

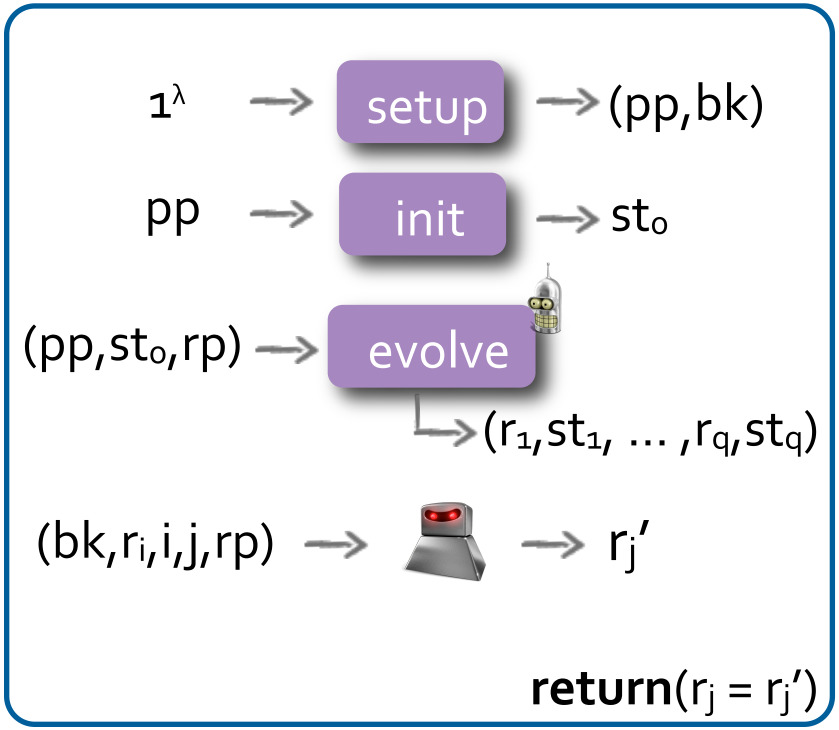
Backdooring models for PRNGs [DPSW16]

We consider a PRNG which we evolve according to a refresh pattern rp, defining a sequence of next and refresh calls.

Game ST-BPRNG(, , i, j, rp)



Game OUT-BPRNG(, , i, j, rp)



A simple backdoored PRNG [DPSW16]

- Dodis et al. (2013) present a construction of a provably robust PRNG
- Crucially, the output is produced by using a forward secure PRG in-between refreshes.
- Simply replace this with a BPRG (and tweak the entropy accumulation process).
- Backdoor attacker can then compromise the PRNG in the period between refreshes.
- But the PRNG is still robust against a normal attacker.
- Challenge: Can we design a backdoored PRNG in which the backdoor attacker can move past refreshes?

Construction of a backdoored PRNG [DPSW16]

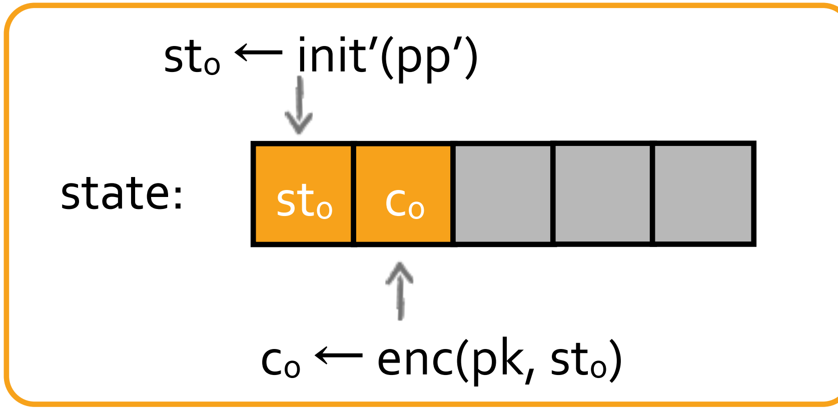
Building blocks

- A robust PRNG′ =(setup′, init′, refresh′, next′)
- An IND\$-CPA secure rerandomizable encryption scheme
  PKE = (keygen, enc, rerand, dec)

setup

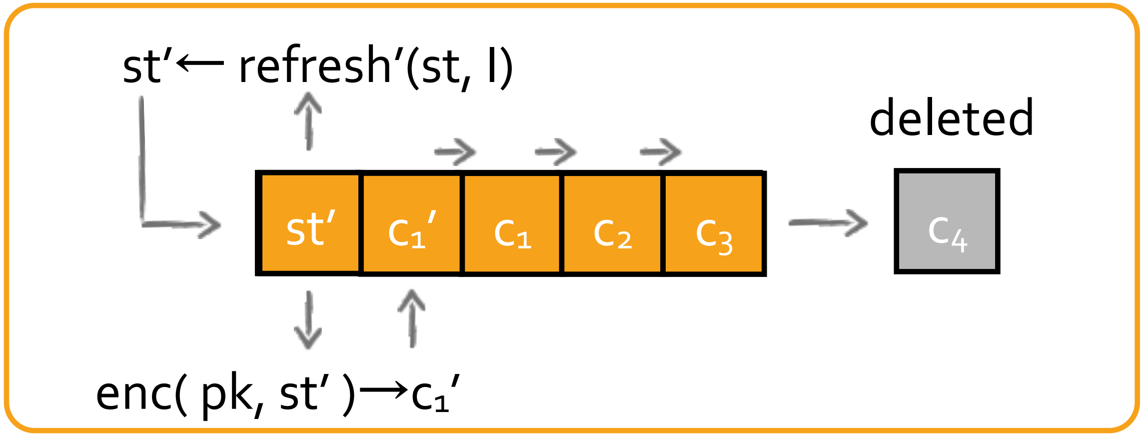
(pk, sk) ← keygen
(pp′, ⊥) ← setup′
pp ← (pp′, pk)
bk ← sk
return (pp, bk)

init

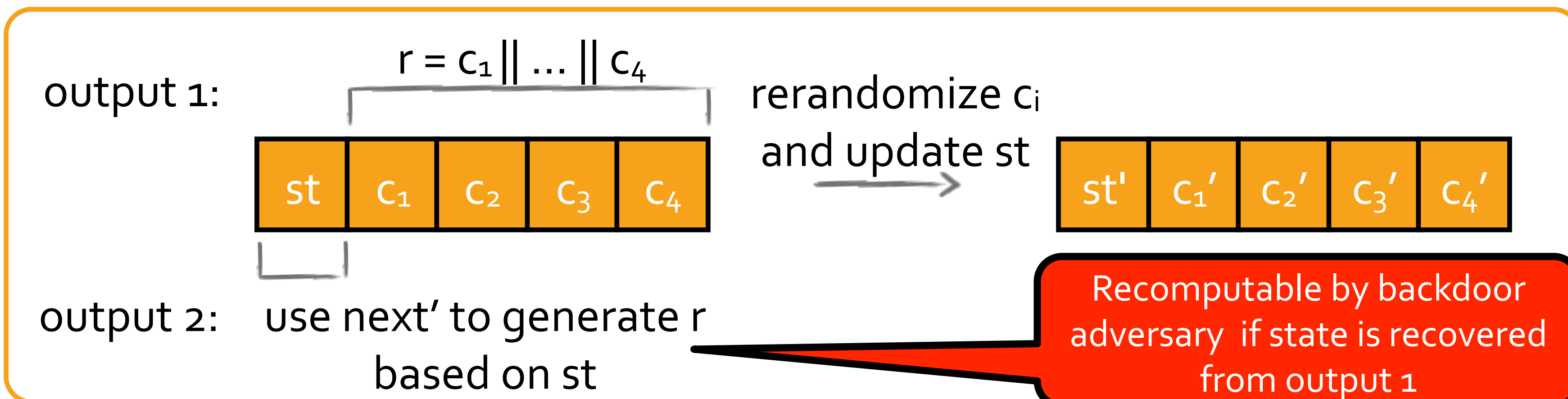


Construction of a backdoored PRNG [DPSW16]

refresh



next



Full construction [DPSW16]

$\overline{\text{setup}}$

$pp \leftarrow\!\!\!\!\leftarrow \text{setup}$

$s \leftarrow \text{refresh}(pp, s, I)$

$\phi \leftarrow 1$

return $(s, C_1 \ldots C_k, \phi)$

$\overline{\text{next}}(\overline{pp}, \overline{S})$

parse $\overline{pp}$ **as** (pp, pk)

$\phi \leftarrow 0$

$\overline{S} \leftarrow (s, C_1 \ldots C_k, \phi)$

return $(\overline{r}, \overline{S})$

$\mathscr{B}(\overline{pp}, bk, \overline{r}_i, i, j, \vec{rp})$

parse $\overline{pp}$ **as** (pp, pk)

Robustness of PRNG = (setup, init, refresh, next) follows from:

- Robustness of PRG' = (setup', init', refresh', next')
- IND\$-CPA security and rerandomizability of PKE = (keygen, enc, rerand, dec)

Advantage of Big Brother in the OUT-BPRNG game is approx. ¼ for i,j values in 'range' and 0 otherwise.

Impossibility result [DPSW16]

Our backdoored PRNG construction crucially relies on storing snapshots of the state, and the degree of backdooring is limited by the size of the state space.

We show that this is inherent to a class of distribution samplers:

For any ε-robust PRNG, any *well-behaved* distribution sampler, any sequence of queries, any legitimate subsequence f, any j and k:

$$H_\infty(\mathbf{S}_{f(j)} \mid R_{f(j)+k}, pp) \geq (j+1)/2 \cdot \log(1/\varepsilon) - \min(l, n)$$

where n is the size of the state, and l is the output size.

Concluding remarks

Concluding remarks

ROYAL HOLLOWAY UNIVERSITY OF LONDON

The bad news:

- Provably forward-secure PRGs can be backdoored in the strongest sense possible: initial state recovery from any single output.
- Provably robust PRNGs can be backdoored to allow Big Brother to recover previous output values, even if the PRNG is refreshed.

The slightly better news:

- BPRGs must look like public key primitives.
- Robust PRNGs provide some resistance against backdooring.

Future work:

- Stronger impossibility results, immunizers for BPRNGs, additional constructions of BPRGs and BPRNGs with more compact state or stronger backdooring,…