Lecture 7 – Randomness, entropy, TRNG/PRNGs, stream ciphers, conspiracy ~~theories~~

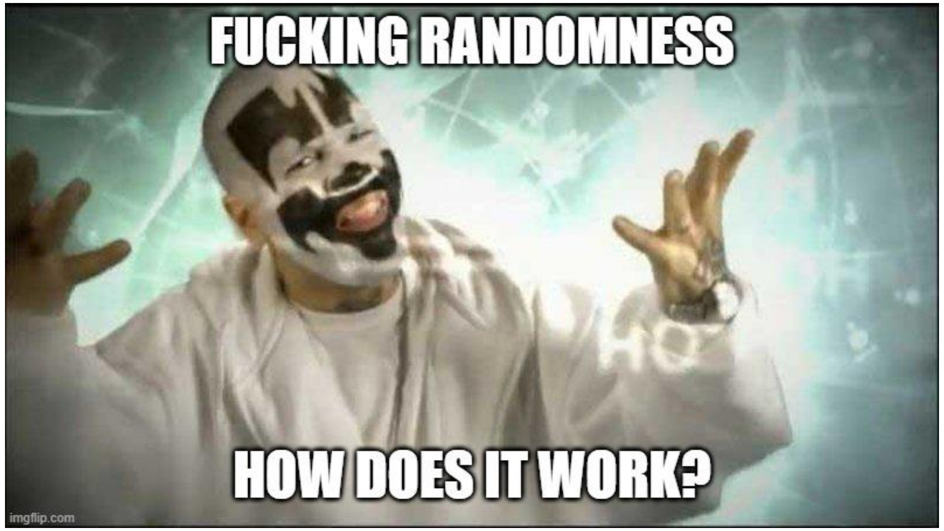
TEK4500

06.10.2021

Håkon Jacobsen

hakon.jacobsen@its.uio.no

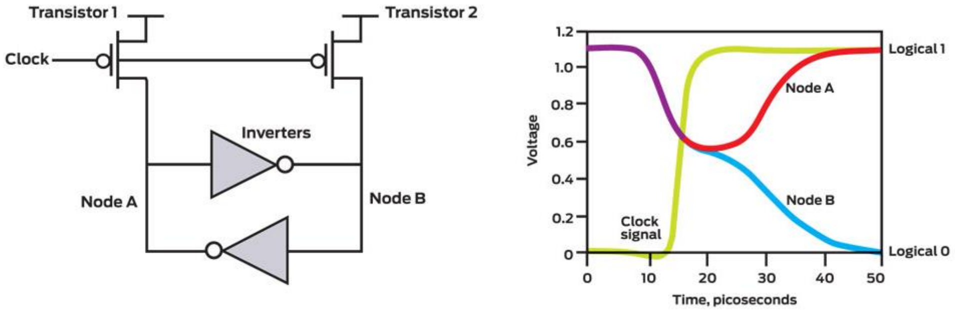
Randomness



TRNG

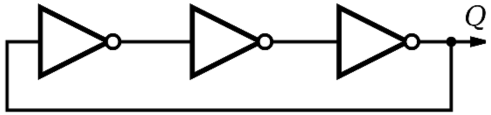
Entropy sources

Thermal noise



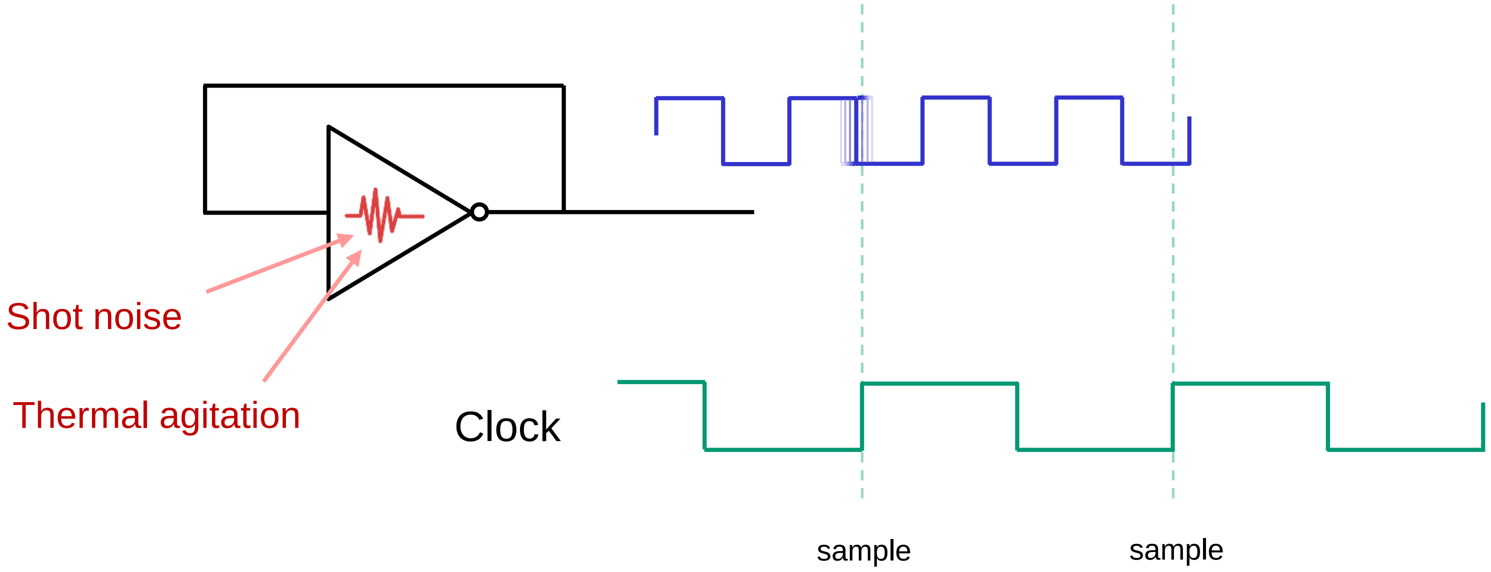
https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/

Ring oscillators

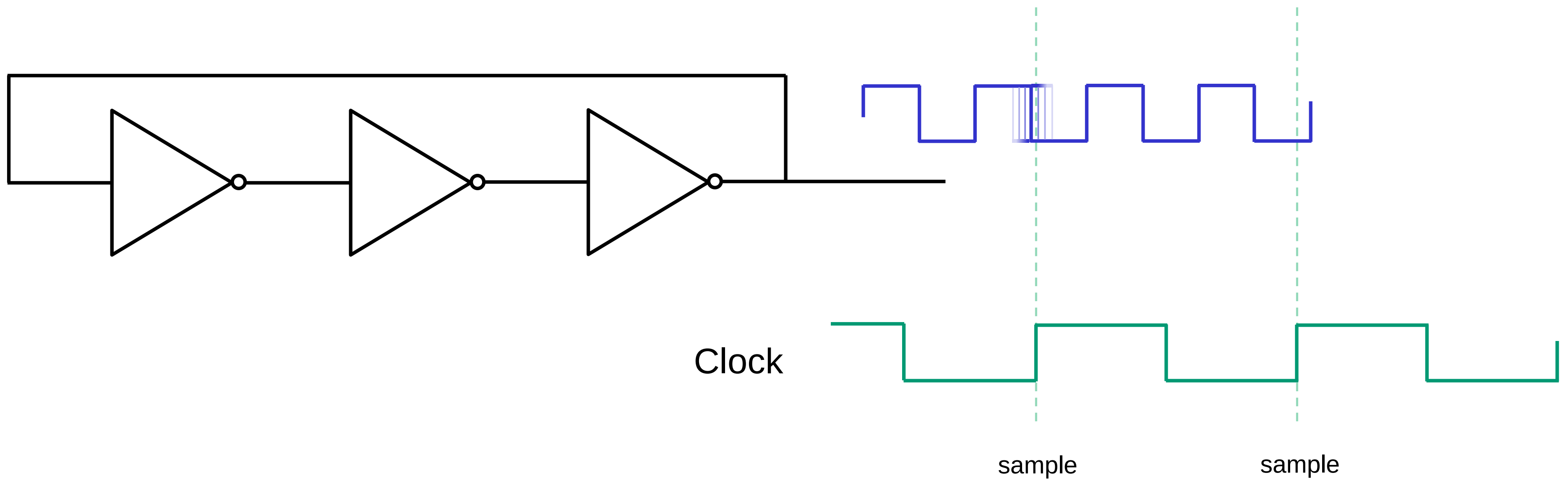


Quantum magic
(radioactive decay, quantum tunneling, etc…)

Ring-oscillators

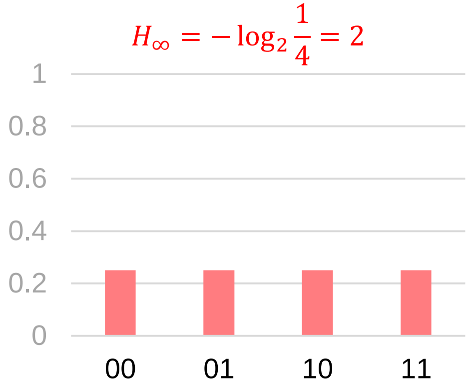


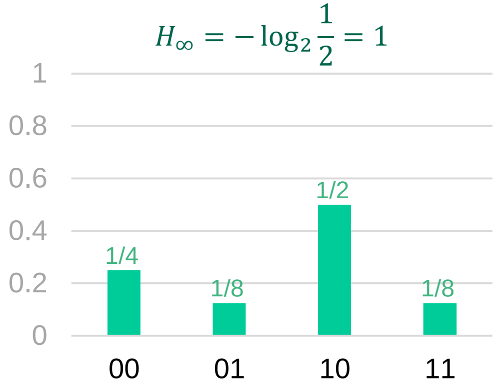
Ring-oscillators

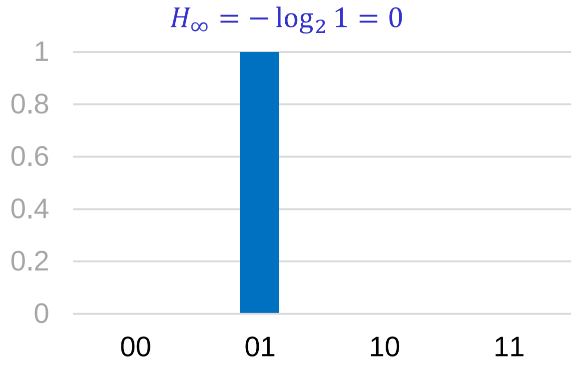


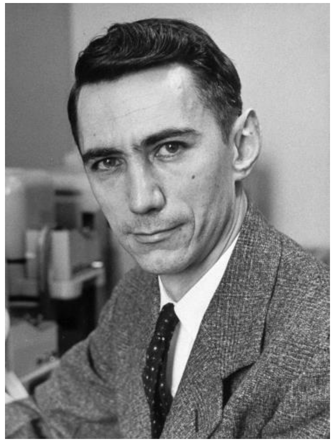
Entropy

- Measure of uncertainty
  - Measured in bits
  - $H_\infty = \text{min-entropy} \stackrel{\text{def}}{=} -\log_2\left(\max_x \Pr[x]\right)$
  - $\Pr[\text{best guessing strategy}] \leq 2^{-H_\infty}$







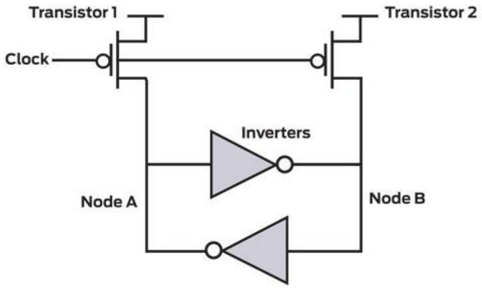


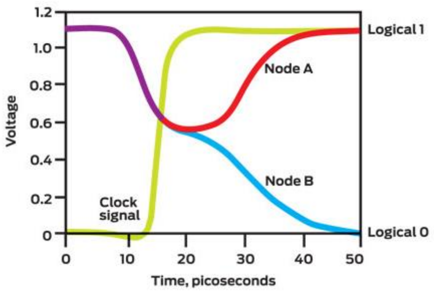
Claude Shannon

- **Examples:**
  - Fair coin: $H_\infty = 1$
  - Fair 6-sided die: $H_\infty = -\log_2 \frac{1}{6} \approx 2.58$
  - Uniform 128-bit string: $H_\infty = 128$
  - Uniform n-bit string + uniform m-bit string: $H_\infty = n + m$

TRNG

Thermal noise

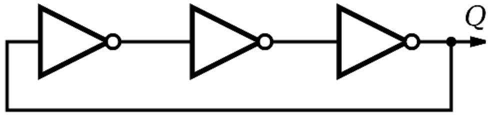




Entropy sources

https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/

Ring oscillators



Quantum magic
(radioactive decay, quantum tunneling, etc…)

Problems with TRNGs

...and slow!

- **Biased sources**
  - Biased bits: $p_0 = 0.25 \quad p_1 = 0.75$
  - Symmetric schemes (PRFs, MACs, encryption schemes, etc.) require *uniform* keys
  - De-bias (von Neumann): create *two* bits; $01 \mapsto 0,\ 10 \mapsto 1,\ 00/11 \mapsto \text{try again}$ (in practice: hash with SHA2-256)

  - **Example:**

    `password = lxiqlxptnpwhraxvfrdgubgfvhjx` (28 random lower-case letters ASCII encoded)

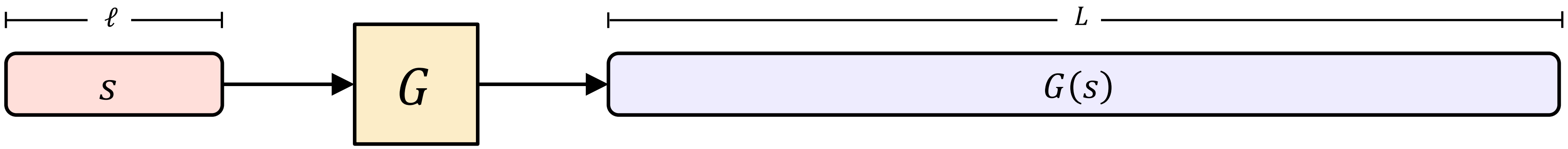
    - $|\mathcal{PW}| = 26^{28} > 2^{131}$ (i.e., password min-entropy ≈ 131 bits)
    - Bit-length password = 28 · 8 = 224 bits
    - AES-128 key: `key = bytes(password[0:15])`
    - What's the min-entropy of key?
      - Each byte is between 0x61 ('a') and 0x7a ('z') ⟹ 4 top bits always starts with 0110 or 0111!
      - min-entropy $\approx 16 \cdot 4.7 = 75.2$ bits!

- **Correlated sources**
  - Value of bit 73 may depend on bit 5
  - Symmetric schemes (PRFs, MACs, encryption schemes, etc.) require *independent* keys
  - De-correlate: much more difficult! (in practice: hash with SHA2-256)

Pseudorandom generators (PRG) – syntax

Have: a short string s in $\{0,1\}^\ell$ (**uniformly** and **independently** distributed)
Want: a *long* string S in $\{0,1\}^L$ (uniformly and independently distributed)

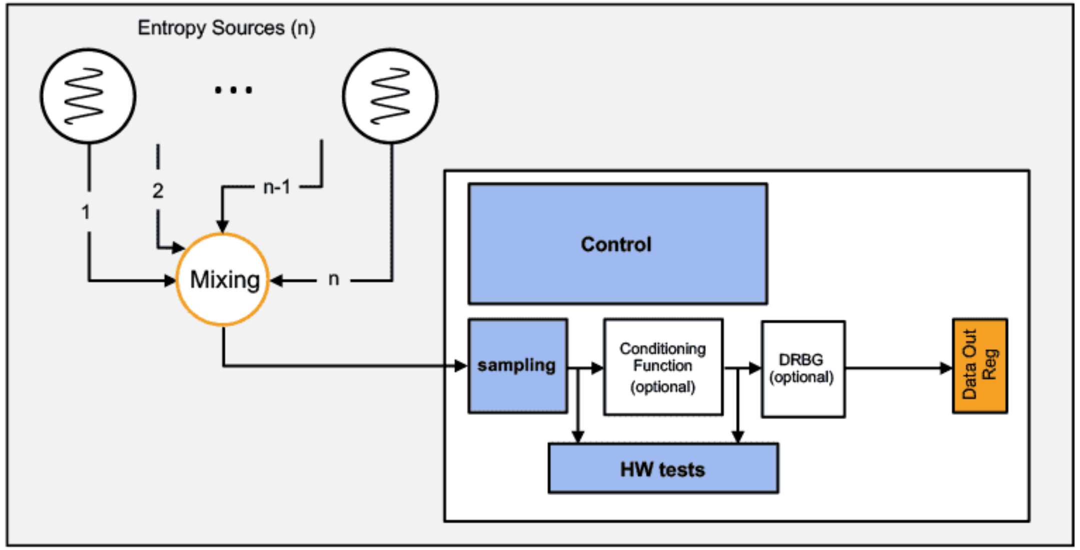
Solution: a **pseudorandom generator (PRG)**, i.e. a function $G : \{0,1\}^\ell \rightarrow \{0,1\}^L$



- Expansion: $L \gg \ell$
- Pseudorandomness: $G(s)$ should look like a truly random string $U \in \{0,1\}^L$

Random generators

- Common design:
  - TRNG generates short random seed
  - PRNG expands seed to “infinite” length



- Examples:
  - /dev/urandom
  - CryptGenRandom
  - Intel RDRAND
- Debian OpenSSL RNG bug
  - `// MD_Update(&m,buf,j);`
  - Only 32,767 possibilities for seed ≈ 15 bits of entropy

Statistical tests

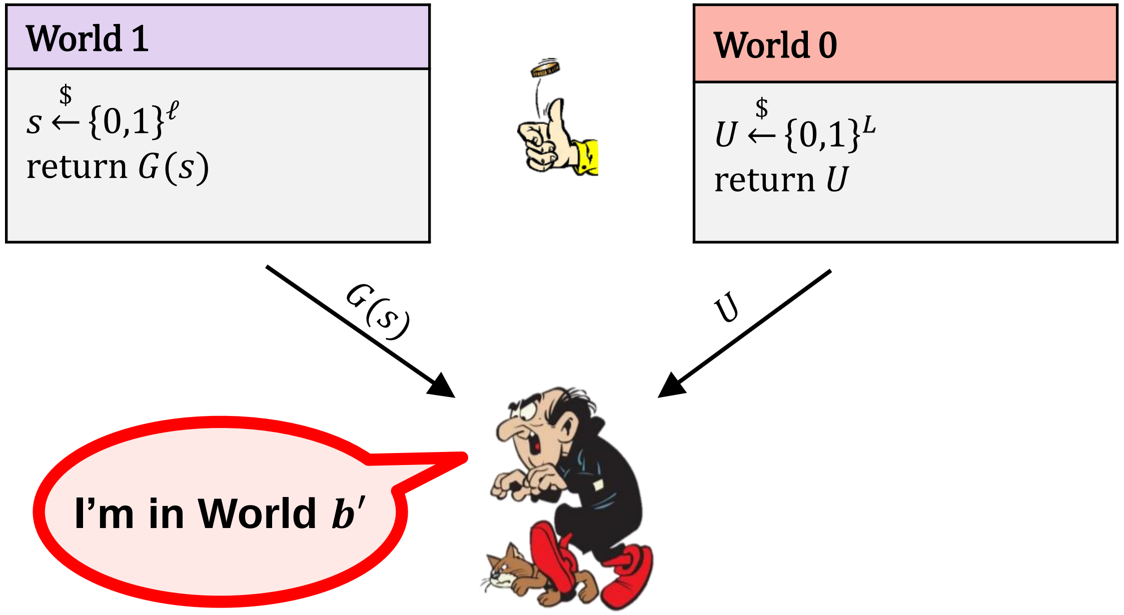
1010100111010110001001000111110000101010000000001010011111

- Is this a random string? Our answer: question not valid!
- What does that even mean?
- Suggestions:
  - A random string should have roughly 50% zeros and ones (how much can you deviate?)
  - A continuous run of zeros (or ones) shouldn't be too long (how long?)
  - ≈ 25% of 2-bit substrings should be 00, 25% should be 01, …
  - ≈ 12.5% of 3-bit substrings should be 000, 12.5% should be 001, ….
  - A random string should not be compressible (related to Kolmogorov-complexity)
  - …

PRNG – security definition

$\mathbf{Exp}_G^{\mathrm{prg}}(A)$

$b \xleftarrow{\$} \{0,1\}$
$s \xleftarrow{\$} \{0,1\}^{\ell}$
$U_1 \leftarrow G(s)$
$U_0 \xleftarrow{\$} \{0,1\}^{L}$
$b' \leftarrow A(U_b)$
return $b' \stackrel{?}{=} b$

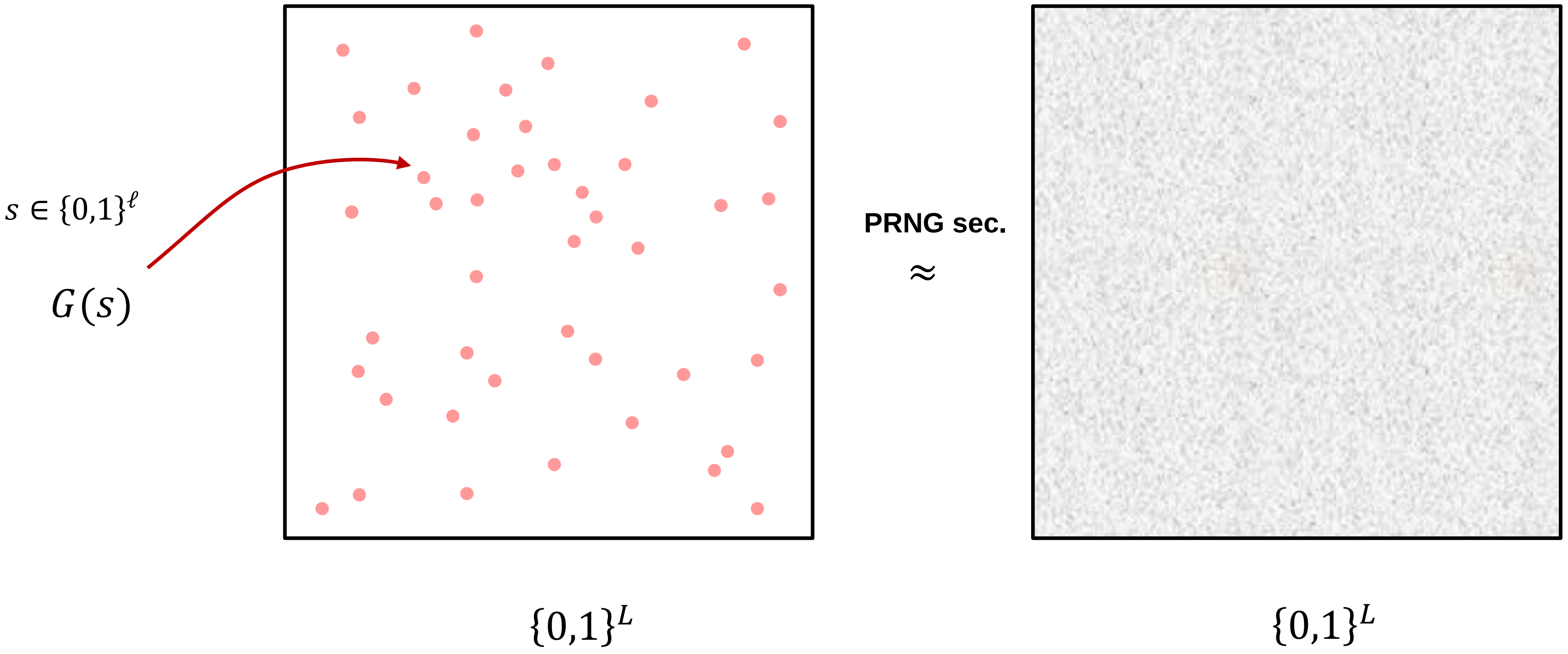


Definition: The **PRNG advantage** of an adversary A is

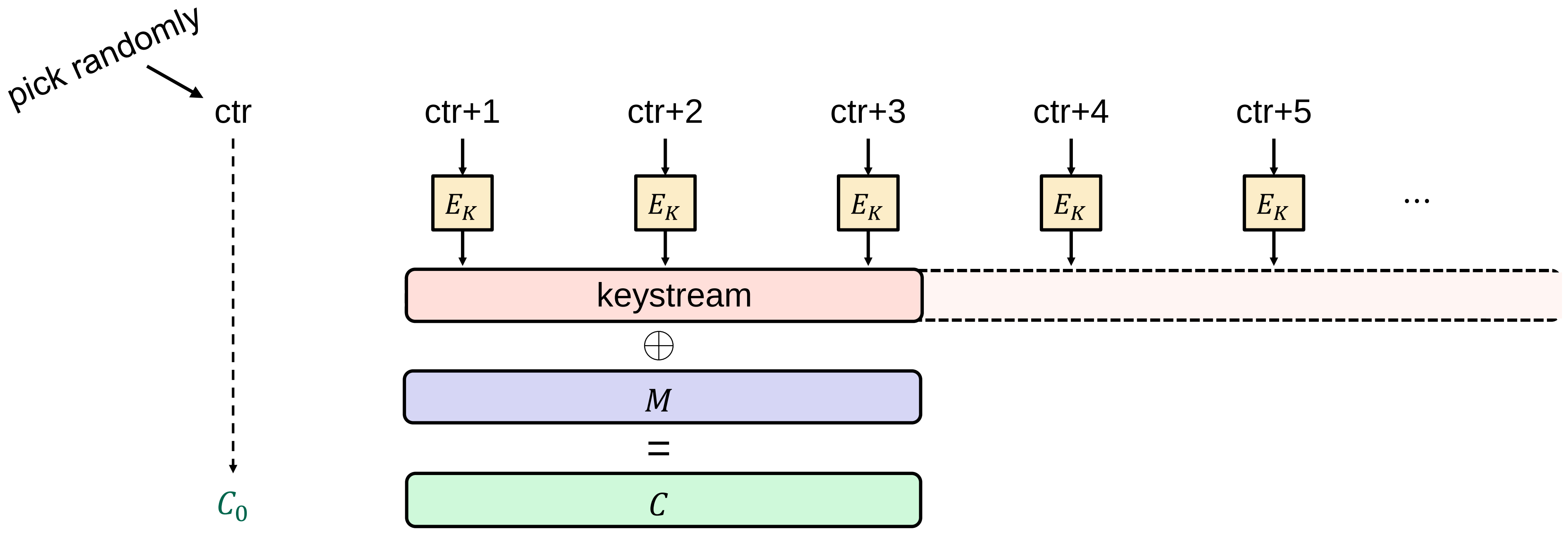
$$\mathbf{Adv}_G^{\mathrm{prg}}(A) = \left|2 \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{prg}}(A) \Rightarrow \mathrm{true}\right] - 1\right|$$

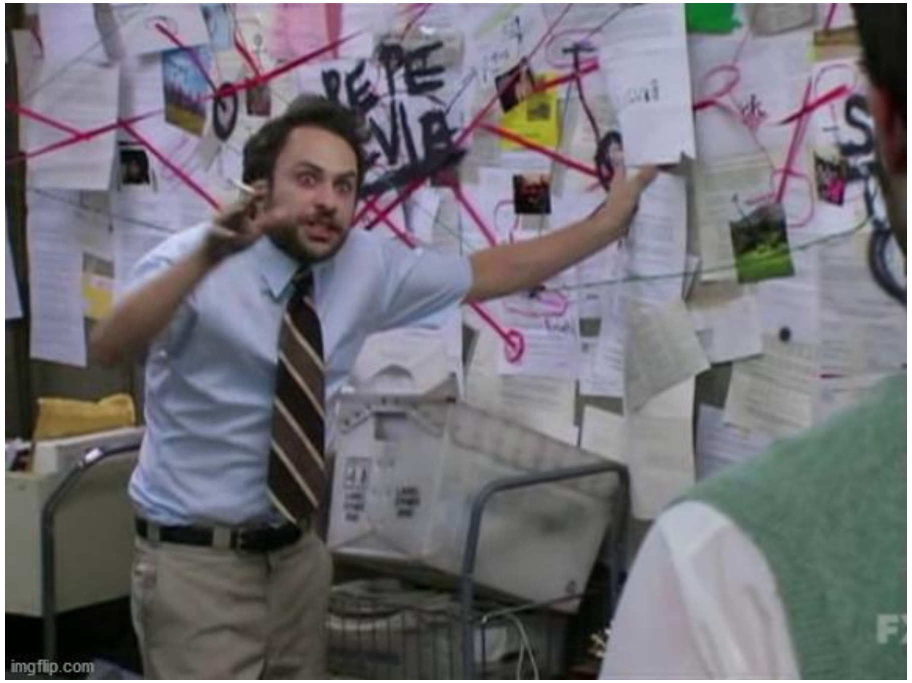
$$2 \cdot \Pr[b' = b] - 1 = \Pr[\, b' = 1 \mid b = 1\,] + \Pr[\, b' = 0 \mid b = 0\,] - 1$$
$$= \Pr[\, b' = 1 \mid b = 1\,] - \Pr[\, b' = 1 \mid b = 0\,]$$
$$= \Pr\left[A\big(G(s)\big) \Rightarrow 1\right] - \Pr[A(U) \Rightarrow 1]$$

Pseudorandomness



Creating PRNGs – CTR-mode





AFTER THE BREAK…

Juniper Networks

- Juniper Networks: big manufacturer of network equipment (routers, VPNs, firewalls, etc.)
  - Major customers: telcos, banks, US DoD
- **2015:**

IMPORTANT JUNIPER SECURITY ANNOUNCEMENT

During a recent internal code review, Juniper discovered unauthorized code in ScreenOS that could allow a knowledgeable attacker to gain administrative access to NetScreen® devices and to decrypt VPN connections.

- Hackers had obtained access to source code repository
- Only change:

```
--- Qx = 2c55e5e45edf713dc43475effe8813a60326a64d9ba3d2e39cb639b0f3b0ad10
+++ Qx = 9585320eeaf81044f20d55030a035b11bece81c785e6c933e4a8a131f6578107
```

PRNG standardization

NIST Special Publication 800-90

Recommendation for Random Number Generation Using Deterministic Random Bit Generators

Elaine Barker
John Kelsey

Computer Security Division
Inforamtion Technology Laboratory

C O M P U T E R   S E C U R I T Y

June 2006

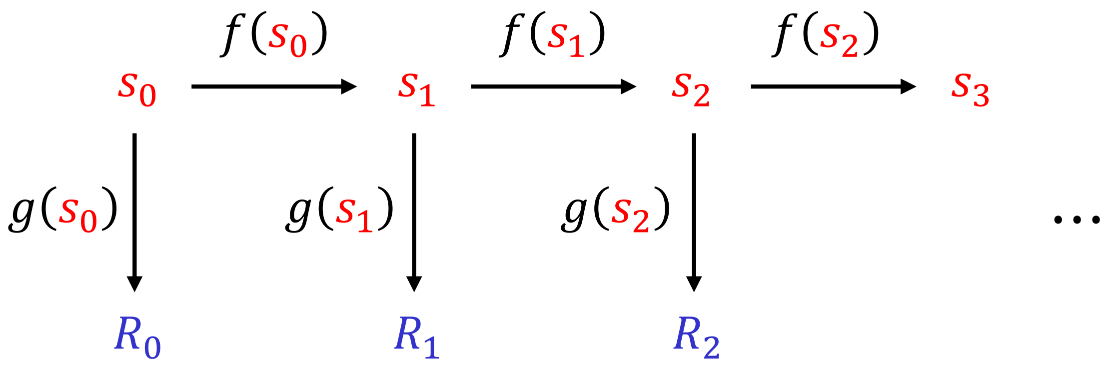
U.S. Department of Commerce
Carlos M. Gutierrez, Secretary

Technology Administration
Robert Cresanti, Under Secretary of Commerce for Technology

National Institute of Standards and Technology
William Jeffrey, Director

NIST National Institute of Standards and Technology • Technology Administration • U.S. Department of Commerce

PRNG

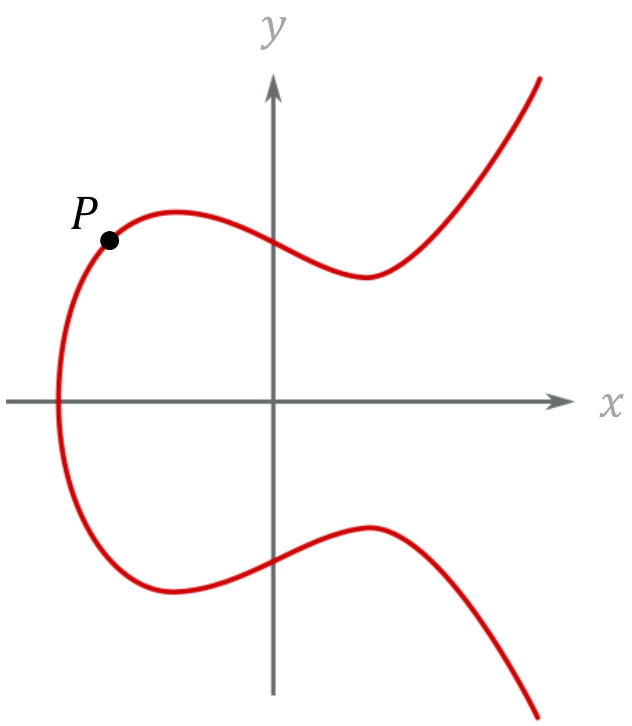


Elliptic curves 101

- $P = (x, y)$ point on elliptic curve
- x and y are 32-byte integers
- Points P and Q can be added to get another point $P + Q$
- Special case: add P to itself n times

$$nP = P + P + \cdots + P$$

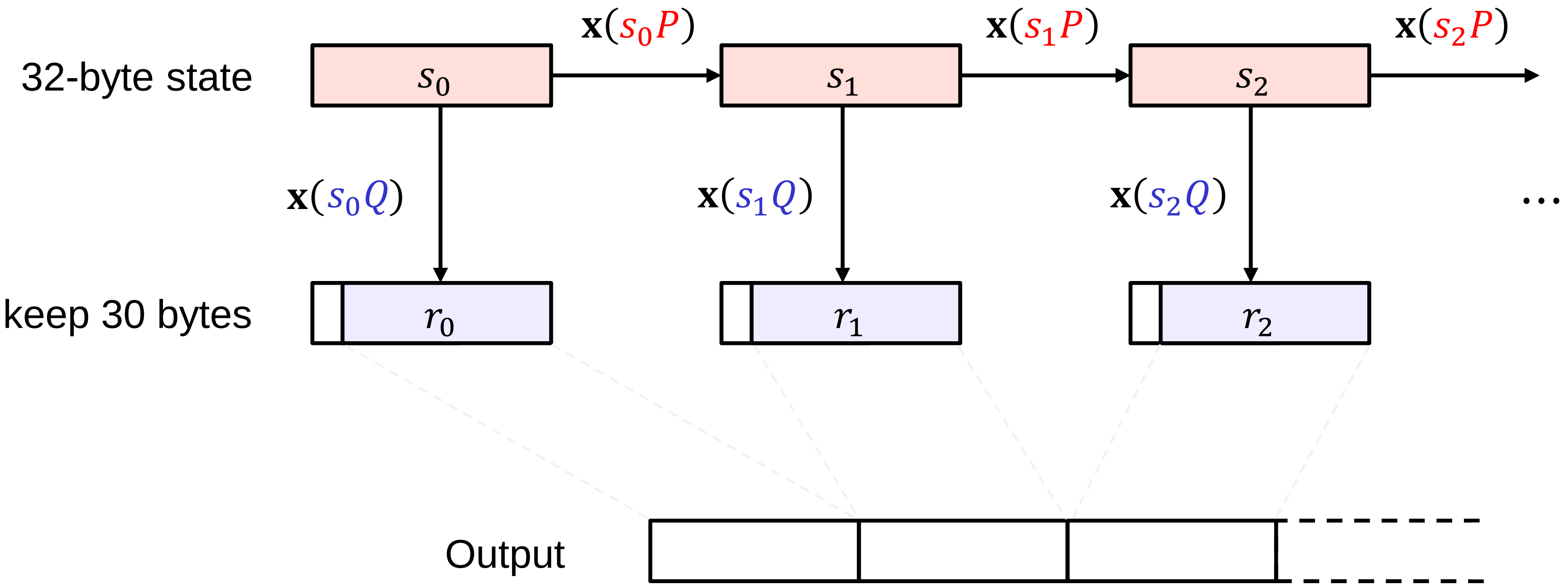
- Fact: given P and nP for **secret** n, hard to find n



Dual EC DRBG

P, Q: **public** points on an elliptic curve

$\mathbf{x}(\cdot)$: x-coordinate (32 bytes)



Where does Q (and P) come from?

```
Subject: RE: Minding our Ps and Qs in Dual_EC
From: "Don Johnson" <DJohnson@cygnacom.com>
Date: Wed, October 27, 2004 11:42 am
To: "John Kelsey" <john.kelsey@nist.gov>
John,

P=G.

Q is (in essence) the public key for some random private key.

It could also be generated like a(nother) canonical G, but NSA kyboshed
this idea, and I was not allowed to publicly discuss it, just in case you
may think of going there.

Don B. Johnson

-----Original Message-----

From: John Kelsey [mailto:john.kelsey@nist.gov]
Sent: Wednesday, October 27, 2004 11:17 AM
To: Don Johnson
Subject: Minding our Ps and Qs in Dual_EC

Do you know where Q comes from in Dual_EC_DRBG?

Thanks,
-Joh
```


Appendix A: (Normative) Application-Specific Constants

A.1 Constants for the Dual_EC_DRBG

The **Dual_EC_DRBG** requires the specifications of an elliptic curve and two points on the elliptic curve. One of the following NIST approved curves with associated points **shall** be used in applications requiring certification under FIPS 140-2. More details about these curves may be found in FIPS PUB 186-3, the Digital Signature Standard.

Each of following curves is given by the equation:

$$y^2 = x^3 - 3x + b \pmod{p}$$

Notation:

p - Order of the field F_p , given in decimal

r - order of the Elliptic Curve Group, in decimal . Note that r is used here for consistency with FIPS 186-3 but is referred to as n in the description of the **Dual_EC_DRBG.**

a – (-3) in the above equation

b - coefficient above

The x and y coordinates of the base point, i.e., generator G, are the same as for the point P.

A.1.1 Curve P-256

p = 11579208921035624876269744694940757353008614\
34152903141955336313088670978539 51

r = 11579208921035624876269744694940757352999695\
522413576034242225906106851204436 9

b = 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b

Px = 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296

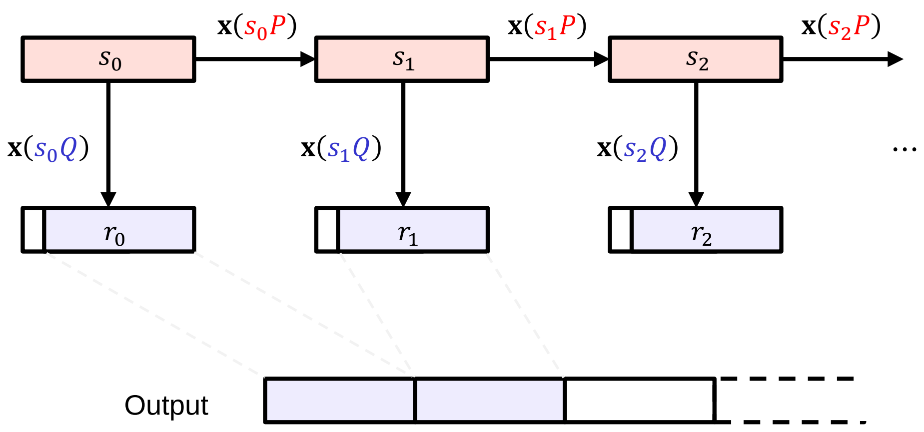
Py = 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5

Qx = c97445f4 5cdef9f0 d3e05e1e 585fc297 235b82b5 be8ff3ef ca67c598 52018192

Qy = b28ef557 ba31dfcb dd21ac46 e2a91e3c 304f44cb 87058ada 2cb81515 1e610046

Dual EC DBRG – something's fishy

- Dual EC is *slow*
  - Orders of magnitude slower than HMAC/AES-CTR based alternatives



- **2006** – Kristian Gjøsteen: Dual EC is *not* a good PRNG
  - Can distinguish output from random with $\mathbf{Adv}^{\mathrm{prg}}_{\mathrm{DualEC}}(KG) \approx 0.0011$
  - Slightly improved by Schoenmakers and Sidorenko

- **2007** – Shumow and Ferguson: Dual EC can be *backdoored*
  - What if $P = dQ$ for a *secret* d only you know?
  - If you know full $s_1 Q$, compute $d(s_1 Q) = s_1(dQ) = s_1 P = s_2$
  - Because of truncation need to guess 2 top bytes ($\approx 2^{16}$ additional work)

- **2007** – NIST adds appendix to standard on how to create P and Q yourself
  - Continues to recommend existing P and Q

- Most cryptographers: who cares? No one is going to use Dual EC anyway …

- **2013** – Edward Snowden leak: a project called Bullrun exists within the NSA
  - Purpose: "*Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.*"
  - Turns out Juniper Networks made Dual EC their PRNG in ScreenOS from 2008

Juniper PRNG

ScreenOS does make use of the Dual_EC_DRBG standard, but is designed to not use Dual_EC_DBRG as its primary random number generator. ScreenOS uses it in a way that should not be vulnerable to the possible issue that has been brought to light. Instead of using the NIST recommended curve points it uses self-generated basis points and then takes the output as an input to FIPS/ANSI X.9.31 PRNG, which is the random number generator used in ScreenOS cryptographic operations.

Juniper Knowledge Base Article KB28205

Juniper PRNG

ScreenOS v.6.1

```c
char block[8], seed[8], key[24];          // X9.31 vars

unsigned int index, calls_since_reseed;

void prng_reseed(void) {


    // same as v6.2


 }

void prng_generate(char *output) {
   unsigned int index = 0;
   calls_since_reseed++;
   if (calls_since_reseed > 10 000)
     prng_reseed();
   for (; index < 20; index += 8) {
     ...
     x9_31_generate_block(seed, key, block);
     ...
     memcpy(&output[index], block, 8);
   }
}
```

caller-supplied buffer

private variable

only 20 bytes

ScreenOS v.6.2

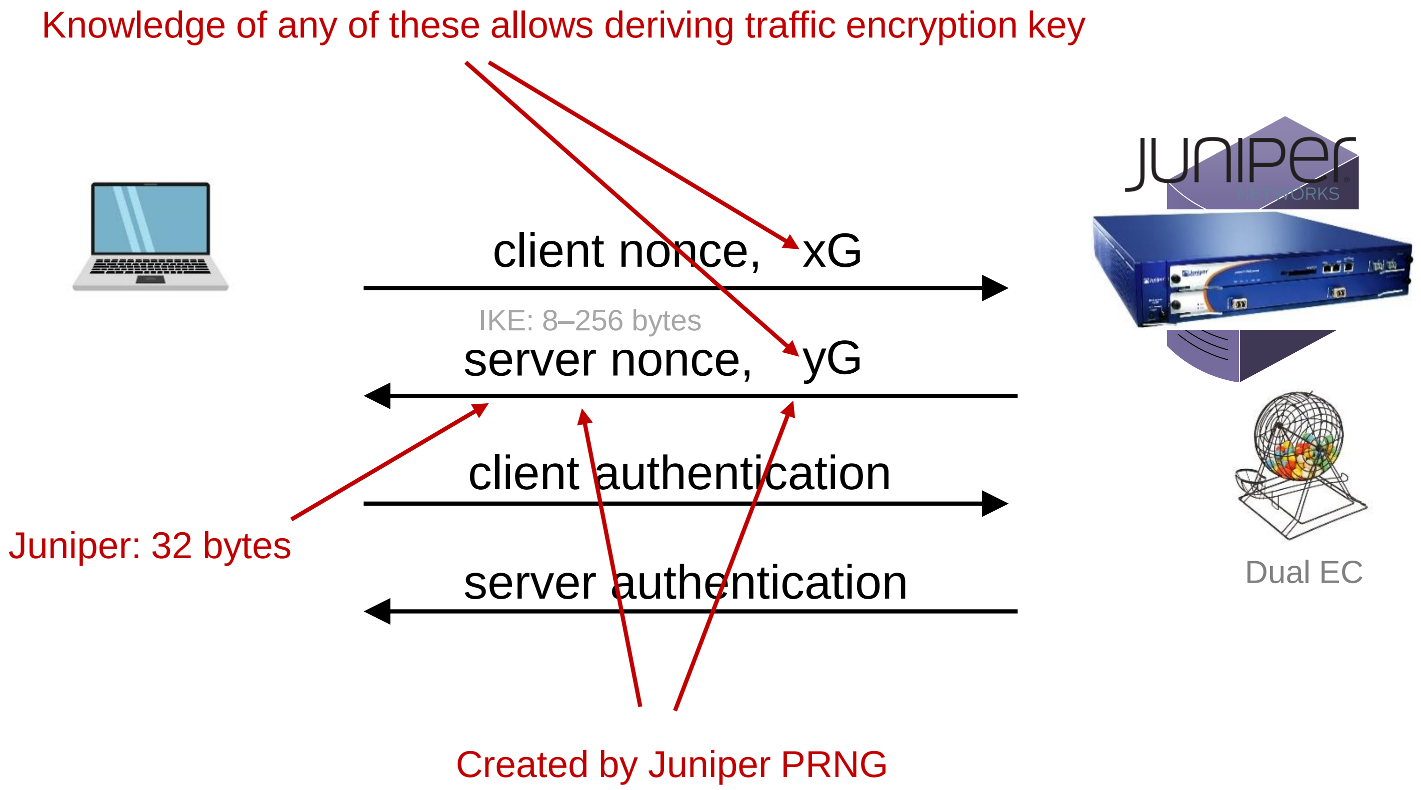
```c
char block[8], seed[8], key[24];     // X9.31 vars
char output[32];                     // prng_generate output
unsigned int index;

void prng_reseed(void) {
   dual_ec_generate(output, 32);
   memcpy(seed, output, 8);
   index = 8;
   memcpy(key, &output[index], 24);
   index = 32;
}


void prng_generate(void) {
   index = 0;

   prng_reseed();

   for (; index < 32; index += 8) {
     ...
     x9_31_generate_block(seed, key, block);
     ...
     memcpy(&output[index], block, 8);
   }
}
```

global variable

global shared buffer

Source: *Where did I leave my keys? Lessons from the Juniper Dual EC incident* https://dl.acm.org/doi/10.1145/3266291

IKEv2 / IPsec



Knowledge of any of these allows deriving traffic encryption key
client nonce, xG
IKE: 8–256 bytes
server nonce, yG
client authentication
server authentication
Juniper: 32 bytes
Created by Juniper PRNG
JUNIPER
Dual EC

Juniper Networks backdoor

- Juniper Networks: big manufacturer of network equipment (routers, VPNs, firewalls, etc.)
  - Major customers: telcos, banks, US DoD
- **2015:**

IMPORTANT JUNIPER SECURITY ANNOUNCEMENT

During a recent internal code review, Juniper discovered unauthorized code in ScreenOS that could allow a knowledgeable attacker to gain administrative access to NetScreen® devices and to decrypt VPN connections.

- Hackers had obtained access to source code repository
- Only change:

```
--- Qx = 2c55e5e45edf713dc43475effe8813a60326a64d9ba3d2e39cb639b0f3b0ad10
+++ Qx = 9585320eeaf81044f20d55030a035b11bece81c785e6c933e4a8a131f6578107
```

Juniper created Q (yay! No NSA)

Who dis?

Juniper Networks backdoor

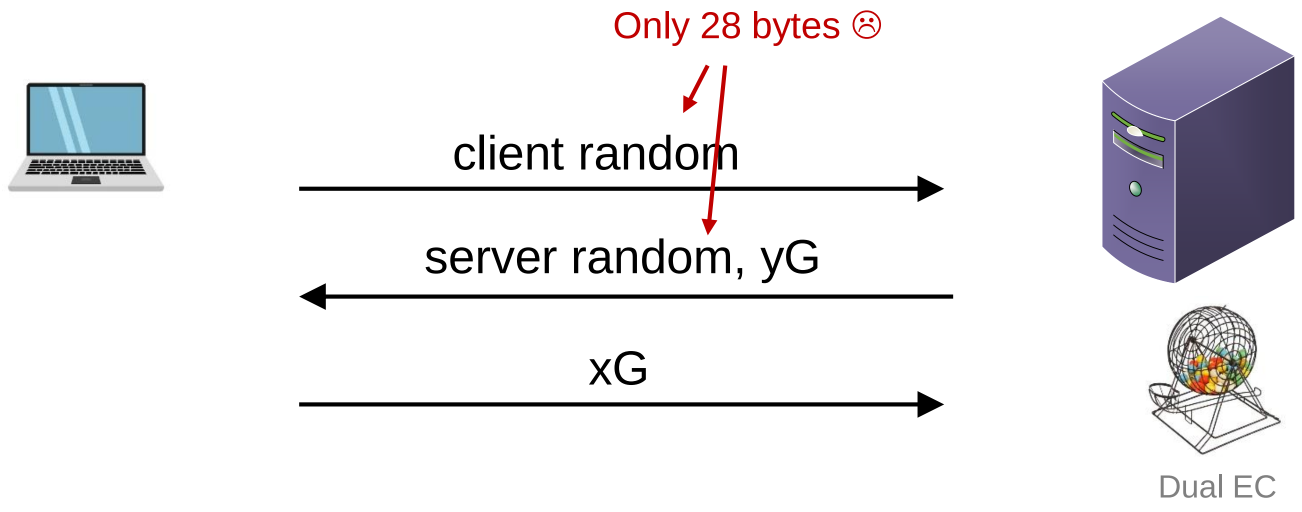
- **2008** – Juniper starts using Dual EC in ScreenOS

- **2012** – Someone hacks into Juniper's code repositories
  - Changes Q point in Dual EC

- **2015** – Juniper discovers intrusion
  - Changes Q back to its original value

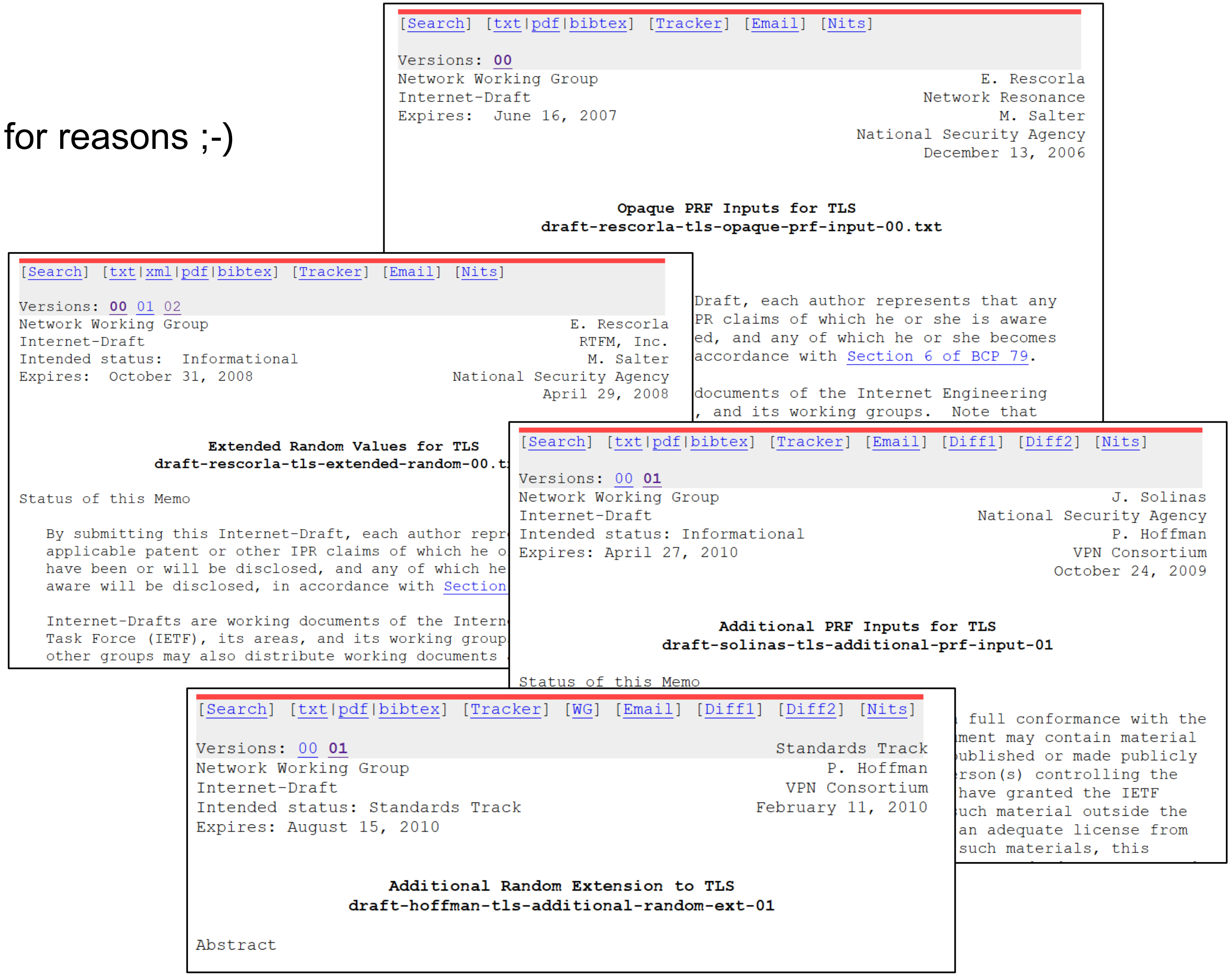
JUNIPer NETWORKS®

TLS



Extended Random

- NSA: please make the TLS nonces bigger…for reasons ;-)
- Implementing Extended Random makes exploiting Dual EC 10,000 times easier
- No real cryptographic justification exists for making them longer


[Search] [txt|pdf|bibtex] [Tracker] [Email] [Nits]

Versions: 00

Network Working Group E. Rescorla
Internet-Draft Network Resonance
Expires: June 16, 2007 M. Salter
National Security Agency
December 13, 2006

Opaque PRF Inputs for TLS
draft-rescorla-tls-opaque-prf-input-00.txt

[Search] [txt|xml|pdf|bibtex] [Tracker] [Email] [Nits]

Versions: 00 01 02

Network Working Group E. Rescorla
Internet-Draft RTFM, Inc.
Intended status: Informational M. Salter
Expires: October 31, 2008 National Security Agency
April 29, 2008

Extended Random Values for TLS
draft-rescorla-tls-extended-random-00.t

Status of this Memo

By submitting this Internet-Draft, each author repr
applicable patent or other IPR claims of which he o
have been or will be disclosed, and any of which he
aware will be disclosed, in accordance with Section

Internet-Drafts are working documents of the Intern
Task Force (IETF), its areas, and its working group
other groups may also distribute working documents

[Search] [txt|pdf|bibtex] [Tracker] [Email] [Diff1] [Diff2] [Nits]

Versions: 00 01

Network Working Group J. Solinas
Internet-Draft National Security Agency
Intended status: Informational P. Hoffman
Expires: April 27, 2010 VPN Consortium
October 24, 2009

Additional PRF Inputs for TLS
draft-solinas-tls-additional-prf-input-01

Status of this Memo

[Search] [txt|pdf|bibtex] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: 00 01 Standards Track

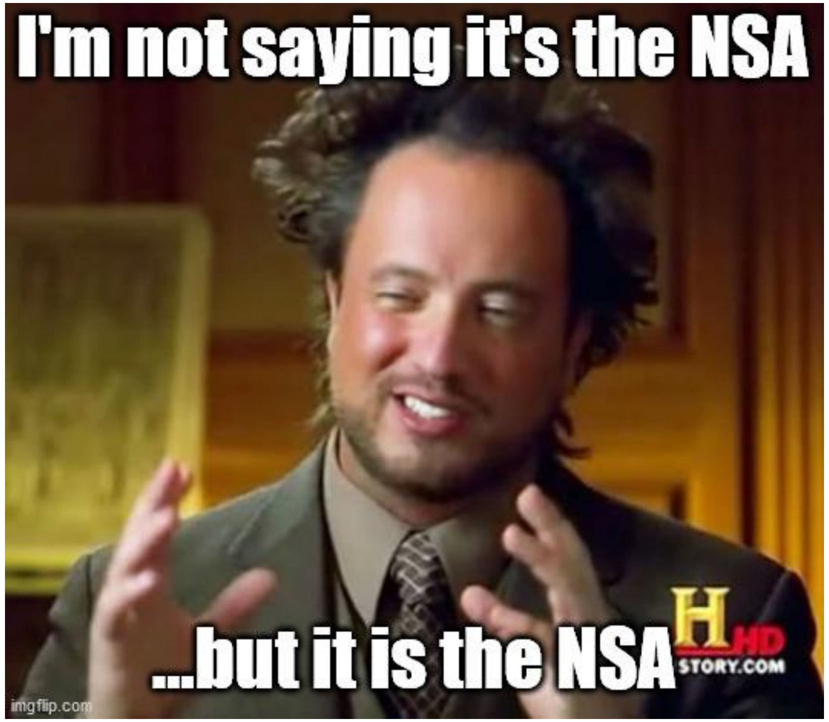
Network Working Group P. Hoffman
Internet-Draft VPN Consortium
Intended status: Standards Track February 11, 2010
Expires: August 15, 2010

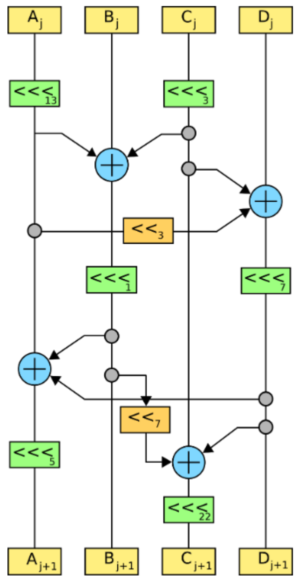
Additional Random Extension to TLS
draft-hoffman-tls-additional-random-ext-01

Abstract

RSA Security – BSAFE

- Turns out Juniper weren't the only one using Dual EC
- RSA Security
  - Big computer and network security company
  - Creator of BSAFE cryptographic library
- **2004** – accepted $10 million from the NSA in order to make Dual EC the default in BSAFE
- **2014** – adapted the TLS Extended Random extension


I'm not saying it's the NSA
...but it is the NSA
HISTORY.COM HD
imgflip.com



END OF PART 1
(SYMMETRIC CRYPTO)

Summary of symmetric cryptography

Primitive	Functionality + syntax	Security goal	Acronym	Examples
Pseudorandom function	Keyed function mapping fixed-length input to fixed-length output $F : \mathcal{K} \times \{0,1\}^{\text{in}} \to \{0,1\}^{\text{out}}$	Indistinguishability from random function	PRF	AES HMAC
Block cipher / pseudorandom permutation	Encrypt fixed-length block $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$	Indistinguishability from random permutation	PRP	AES
Encryption	Encrypt variable-length input $\text{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ $\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \to \mathcal{C}$ (nonce-based)	Confidentiality: attacker should learn nothing about plaintext (except length) from ciphertexts	IND-CPA IND-CCA	CTR CBC$
MAC	Produce fixed-length tag on variable-length message $\text{Tag} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ $\text{Vrfy} : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{\text{Valid}, \text{Invalid}\}$	Integrity: attacker shouldn't be able to forge messages, i.e., create new messages with valid tags	UF-CMA	CBC-MAC CMAC HMAC
Authenticated encryption	Encrypt variable-length input $\text{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ $\text{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ With associated data + nonces (AEAD) $\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ $\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$	Confidentiality + ciphertext integrity Confidentiality (message) + ciphertext and AD integrity	AE	EtM GCM OCB CCM
Hash function	Keyless function mapping variable-length messages to fixed-length tags $H : \mathcal{M} \to \mathcal{Y}$ $H : \{0,1\}^* \to \{0,1\}^n$	Collision-resistance + one-wayness		SHA1 SHA2-256 SHA2-512 SHA3

Summary of symmetric cryptography

Primitive	Functionality + syntax	Security goal	Acronym	Examples
Pseudorandom generator	Function mapping short input seed to long (basically infinite) output string $G : \{0,1\}^\ell \to \{0,1\}^L$	Indistinguishability: output $G(s)$ should look like random string in $\{0,1\}^L$	PRNG/PRG	AES-CTR ChaCha20